

Session Communication and Integration

Guoxin Su^{1,3}, Mingsheng Ying^{1,2}, and Chengqi Zhang¹

¹ Centre for Quantum Computation & Intelligent Systems
University of Technology, Sydney

² State Key Laboratory of Intelligent Technology & Systems
Tsinghua University

³ guoxin@it.uts.edu.au

Abstract. The scenario-based specification of a large distributed system is usually naturally decomposed into various modules. The integration of specification modules contrasts to the parallel composition of program components, and includes various ways such as scenario concatenation, choice, and nesting. The recent development of multiparty session types for process calculi provides useful techniques to accommodate the protocol modularisation, by encoding fragments of communication protocols in the usage of private channels for a class of agents. In this paper, we extend forgoing session type theories by enhancing the session integration mechanism. More specifically, we propose a novel synchronous multiparty session type theory, in which sessions are separated into the communicating and integrating levels. Communicating sessions record the message-based communications between multiple agents, whilst integrating sessions describe the integration of communicating ones. A two-level session type system is developed for π -calculus with syntactic primitives for session establishment, and several key properties of the type system are studied. Applying the theory to system description, we show that a channel safety property and a session conformance property can be analysed. Also, to improve the utility of the theory, a process slicing method is used to help identify the violated sessions in the type checking.

1 Introduction

The description of service accesses in protocols has been long considered as a way to improve the interoperability of program components in a complex computing system, and this is the case for various architecture description languages (e.g. Darwin [19], Wright [1], and PADL [2]) and component-based platforms (e.g. Coyote [3] and Appia [20]). Formal validation methods including model checking and static checking are employed to aid the detection of composition mistakes such as deadlocks and race conditions. For large distributed systems, the specification is usually modularised. Studies on session types [14,10] for process calculi in the dialect of π -calculus [24], especially the recent development of multiparty session types [15], provide useful techniques to accommodate the

protocol modularisation. Informally, a session is a unit of message-based communications with a specific purpose. The suitability of session type theories for describing distributed computing includes two aspects:

- Session type theories provide a global descriptive method for protocols, facilitating the protocol design and verification;
- To handle protocol modularisation, type systems in session type theories project fragments of the protocols on the usage of private channels for intended classes of participants.

However, the origin of session type theories assumes the interleaving situation of different independent behavioural threads [13], but in real-life distributed computing, there are often meaningful interplays between a number of sessions. As an example, the following business protocol consists of four sessions between five agents. A broker and two buyers are in the auction session *Auction* (where *Auction* is seen as a global description of the auction protocol). *Auction* is followed by a transaction session between the auction winner, the broker, and the seller. Two alternative transaction protocols are given for the winner to choose: *DTransaction* is a direct protocol, in which the winner directly transfers money to the seller; *STransaction* is a secure protocol, in which the money is transferred via the broker and an extra (sub-)session *EPay* money transaction between the winner, the bank, and the broker is involved. Therefore, the whole business protocol is the integration of four sessions in the intended ways. It is indeed possible to view the protocol as inseparable, but so-doing violates both the natural understanding of the protocol and the gradual procedure of requirement specification.

To improve the session integration mechanism for session type theories, we argue for the merits of separating sessions into two levels. In the present paper, we propose a theory of two-level synchronous multiparty session types, in which communicating sessions specify the end-point communications of multiple components, whilst integrating sessions describe the gluing of communicating sessions by concatenation, choice, interleaving composition, and nesting composition. Compared with the existing studies in this subject, e.g. [4,15,28,8,7], in addition to the separation of session communication and integration, the novelty of our work includes the following aspects. First, we view sessions as a behavioural rather than data-structural approximation of processes. Besides for statically typing processes in a variant of π -calculus, session types are also executable and equipped with intuitive operational semantics. In the forgoing session type theories, sessions as the specification leave out data structures required in the implementation, but based on the operational semantics of sessions, we investigate the behavioural relation between processes and sessions. We demonstrate that, in spite of session modularisation and integration, behaviours of processes, if typed properly, conform to their session specification. Second, the most recent work on session types witnesses a trend to introduce more expressive session constructs and, correspondingly, more syntactic primitives in the underlying calculi, but the ramification of communicating and integrating sessions in our work does not complicate the syntax of the process calculus. Lastly, to improve the utility

Fig. 1 Syntax of the calculus

$P ::= \pi.P$	prefixing	X	variable
$(\nu a)P$	hiding	$[\text{rec } X]P$	recursion
$r : P$	labelling	$P + P$	choice
$\mathbf{0}$	inaction	$P \mid P$	parallel
$\alpha ::= \pi$	action	τ	silence
$\pi ::= \bar{a}_{[2..n]}(\tilde{c})$	invitation	$a?v$	receiving
$a_{[k]}(\tilde{c})$	acceptance	$a!v$	sending

of the theory, we use a process slicing method to help identify the violated sessions in the type checking. The method decomposes a process into parts with respect to sessions in its session specification and compares each part with the role projected from a corresponding session.

The organisation of the remainder of the paper is as follows. In the next section, we present a process calculus with actions for multiparty session establishment. In Sect. 3, we define the syntax and semantics of communicating and integrating sessions, together with methods to project sessions into roles for processes. In Sect. 4, we develop a two-level session type system and study several key properties of the type system. In Sect. 5, we apply the session type theory to system description and analyse a channel safety property and a behavioural property of session conformance. In Sect. 6, we use a process slicing method to facilitate the identification of violated sessions in type checking. In Sect. 7, we discuss the related work to this paper. Finally, we conclude the paper by outlining the future work. More examples and proof details of the theorems are in the Appendix.

2 The Calculus

This section defines a variant of π -calculus. In the next two sessions, a type discipline based on two-level session types is developed for the calculus. Compared with the existing session type literature, the syntax of our calculus is abstract and close to the original presentation of π -calculus. Our intention is to minimise the side techniques (we return to this point in Sect. 7).

The basic sets are a set of *channels* (a, b, c, c'), a set of *messages* or *message types* (u, v, v'), and a set of *participant names* ($p, q, r, 1, 2$, ect.). The syntax of *processes* and *actions* is given in Figure 1. Sessions, which are informally understood as units of interactions, are established by shared channels. The key syntactic primitives for channel establishment are of the forms $\bar{a}_{[2..n]}(\tilde{c})$ and $a_{[k]}(\tilde{c})$, which are due to [15]. These two prefixes are called *session actions* and a is called a *session channel*. $\bar{a}_{[2..n]}(\tilde{c})$ invites participants 2 to n to join in a session whose communicating channels are \tilde{c} , whilst $a_{[k]}(\tilde{c})$ accepts a session invitation. By the operational semantics, when the actions $\bar{a}_{[2..n]}(\tilde{c})$ and $a_{[k]}(\tilde{c})$ (for each

Fig. 2 Structural congruence

$$\begin{array}{lll}
P \mid Q \equiv Q \mid P & P \mid \mathbf{0} \equiv P & (P \mid Q) \mid R \equiv P \mid (Q \mid R) \\
P + Q \equiv Q + P & P + \mathbf{0} \equiv P & (P + Q) + R \equiv P + (Q + R) \\
(\nu a)\mathbf{0} \equiv \mathbf{0} & (\nu a)(\nu b)P \equiv (\nu b)(\nu a)P & (\nu a)P \mid Q \equiv (\nu a)(P \mid Q) \text{ if } a \notin \text{fc}(Q) \\
[\text{rec } X]\mathbf{0} \equiv \mathbf{0} & [\text{rec } X]R \equiv R \text{ if } X \notin \text{fv}(R) & P \equiv Q \text{ if } P =_\alpha Q \\
l' : l : P \equiv l' : P & l : P \mid l : Q \equiv l : (P \mid Q) & l : (\nu a)P \equiv (\nu a)l : P
\end{array}$$

$2 \leq k \leq n$) are triggered synchronously, a session is established via the session channel a and a sequence of fresh communicating channels \tilde{c} are generated (unlike [15], in which the message transport is asynchronous). In $r : P$, r labels P and is seen as the name of P . In some literature, it is also called the location of P [12]. Other syntactic primitives and constructions are standard and from π -calculus.

Binders are a in $(\nu a)P$, \tilde{c} in $\bar{a}_{[2..n]}(\tilde{c}).P$ or $a_{[k]}(\tilde{c}).P$, and X in $[\text{rec } X]P$. Substitution of channels are standard. In particular, $([\text{rec } X]P)\{a/b\} = [\text{rec } X](P\{a/b\})$. $(\nu \tilde{a})P$ stands for $(\nu a_1) \dots (\nu a_n)P$ where $\tilde{a} = a_1, \dots, a_n$. The left-associative law is adopted when presenting multiple \mid or $+$. We assume the bound name convention for processes. Let $\text{fc}(P)$ and $\text{fc}(\alpha)$ denote the set of free channels in P and α , respectively. $\text{fv}(P)$ is the set of free process variables, and $\text{act}(P)$ the set of prefixes in P . Supposing $\tilde{a} = \text{fc}(P)$ and $|\tilde{c}| = |\tilde{a}|$, $P\langle\tilde{c}\rangle$ refers $P\{\tilde{c}/\tilde{a}\}$.

The structural congruence \equiv is the smallest congruent relation on processes that includes the equations in Figure 2. $P =_\alpha Q$ means that P and Q are variants of *alpha-conversion*. Note that we leave out the equi-recursive equation (e.g. $[\text{rec } X]P \equiv P\{[\text{rec } X]P/X\}$) in the structural laws (it is called recursion-free or replication-free structural congruence in some literature [9]). Consequently, we have the decidability of structural congruence.

Lemma 1 *For any given P, Q , it is decidable if $P \equiv Q$.*

If $X \in \text{fv}(P)$, we let $P^{[X]}$ be $[\text{rec } X]P$; otherwise, $P^{[X]}$ is P . $P \sqsubseteq Q$ means $P + R \equiv Q$ for some R , and $P \sqsubset Q$ means $P + R \equiv Q$ for some $R \neq \mathbf{0}$. $P \sqcup Q$ is defined as follows: if $Q \sqsubseteq P$ then $P \sqcup Q = P$; if $P \sqsubset Q$ then $P \sqcup Q = Q$; otherwise, $P \sqcup Q = P + Q$.

The operational semantics are given through a labelled transition system defined by rules in Figure 3. In the session type literature, the semantics of the process calculus is defined as a reduction system instead of a labelled transition one. The advantage of the former over the latter is a simpler presentation. But because one of our purposes in the present paper is to study the behavioural relation between processes and their session types, the standard operational semantics are more suitable to this end. The rules [INV], [ACC], and [SESS] handle the session establishment, and their intuitive meanings have been explained. [LAB] is for process labelling. The rest of the semantic rules are standard.

Let $\text{proc}(P) = \{Q \mid P \xrightarrow{\tilde{\alpha}} Q\}$. P stimulates Q , denoted $P \succ Q$, if there is a relation $\mathcal{S} \subseteq \text{proc}(P) \times \text{proc}(Q)$ such that if $\langle P, Q \rangle \in \mathcal{S}$ and $Q \xrightarrow{\alpha} Q'$ then there is P' such that $P \xrightarrow{\alpha} P'$ and $\langle P', Q' \rangle \in \mathcal{S}$. We use $P \xrightarrow{\alpha}_{\succ} Q$ to mean

Fig. 3 Operational semantics

[SED]	$a!v.P \xrightarrow{a!v} P$	[RCV]	$a?v.P \xrightarrow{a?v} P$
[INV]	$\bar{a}_{[2..n]}(\tilde{c}).P \xrightarrow{\bar{a}_{[2..n]}(\tilde{c})} P$	[ACC]	$a_{[k]}(\tilde{c}).P \xrightarrow{a_{[k]}(\tilde{c}')} P\{\tilde{c}'/\tilde{c}\}$
[PAR]	$\frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$	[SUM]	$\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$
[COM]	$\frac{P \xrightarrow{a!v} P' \quad Q \xrightarrow{a?v} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q}$	[HID]	$\frac{P \xrightarrow{\alpha} P' \quad \text{fc}(\alpha) \neq a}{(\nu a)P \xrightarrow{\alpha} (\nu a)P'}$
[SESS]	$\frac{P_1 \xrightarrow{\bar{a}_{[2..n]}(\tilde{c})} P'_1 \quad P_i \xrightarrow{a_{[i]}(\tilde{c})} P'_i \ (\forall 2 \leq i \leq n)}{P_1 \mid \dots \mid P_n \xrightarrow{\tau} (\nu \tilde{c})(P'_1 \mid \dots \mid P'_n)}$	[LAB]	$\frac{P \xrightarrow{\alpha} P'}{l : P \xrightarrow{\alpha} l : P'}$
[REC]	$\frac{P \xrightarrow{\alpha} P'}{[\text{rec } X]P \xrightarrow{\alpha} P'\{[\text{rec } X]P/X\}}$	[EQV]	$\frac{P \equiv Q \quad Q \xrightarrow{\alpha} Q' \quad Q' \equiv P'}{P \xrightarrow{\alpha} P'}$

that there is R such that $P \xrightarrow{\alpha} R$ and $R \succ Q$. We say P is *deterministic* (up to structural congruence) if for each $Q \in \text{proc}(P)$, $Q \xrightarrow{\alpha} R$ and $Q \xrightarrow{\alpha} R'$ entail $R \equiv R'$.

Examples of agent behaviours We provide a detailed, but informal description of the interactions between the five agents in the example from Introduction, and then formulate their individual behaviours in the calculus. In the upmost level, the whole business protocol is divided into two stages. The first stage is for the auction session and the second one includes two alternative transaction sessions and a possible nested sub-session.

At the auction stage, the broker initiates the auction session with two buyers, i.e. buyer_1 and buyer_2 . For simplicity, we assume that the buyers already know the base price of the auctioned item. After the auction is initiated, buyer_1 (resp. buyer_2) sends its bid to the broker and the protocol reaches a recursive state. In the recursive state, the broker sends a new quote to the other buyer and the protocol proceeds in the following two alternative branches. (a) If the other buyer does not bid (after some amount of time), then the broker issues an invoice to buyer_1 (resp. buyer_2), finishing the auction. (b) If the other buyer bids, then the broker forwards the latest quote to buyer_1 (resp. buyer_2) and, again, the protocol has two sub-branches: (b1) if buyer_1 (resp. buyer_2) continues to bid, then the protocol returns to the recursive state; (b2) otherwise, the broker issues buyer_2 (resp. buyer_1) an invoice to finish the auction.

At the transaction stage, the buyer that won the auction initiates one of the following two transaction options. (a) If the direct transaction is chosen, then the broker forwards the price to the seller, and the buyer makes the payment to the seller and receives the ordering information. (b) If the secure transaction is chosen, an extra bank transfer session is involved. The buyer authorises the bank to transfer an intended amount of money to the broker. The broker holds the money but informs the seller that the pre-payment is ready, and seller

sends the ordering information to the buyer. After receiving the item, the buyer sends a confirmation message to the broker and the broker finalises the deal by transferring the pre-payment to the seller.

The formal description of the broker's behaviour is given by the following process.

$$\begin{aligned}
P_{\text{broker}} &\stackrel{\text{def}}{=} \overline{\text{auc}}_{[2..3]}(a_{1,2}, a_{1,3}). \sum_{i \in \{1,2\}} (a_{1,i+1}? \text{bid}. [\text{rec } X](a_{1,4-i}! \text{quote}. \\
&\quad (a_{1,i+1}! \text{invoice}. P_{\text{broker}}^i + a_{1,4-i}? \text{bid}. a_{1,i+1}! \text{quote}. \\
&\quad (a_{1,i+1}? \text{bid}. X + a_{1,4-i}! \text{invoice}. P_{\text{broker}}^{3-i})))) \\
P_{\text{broker}}^i &\stackrel{\text{def}}{=} \text{dTran}_{[2]}^i(b_{1,2}, b_{2,3}). b_{2,3}! \text{price}. \mathbf{0} + \text{sTran}_{[2]}^i(c_{1,2}, c_{1,3}, c_{2,3}). \\
&\quad \text{epay}_{[2]}^i(d_{1,3}, d_{2,3}). d_{1,3}? \text{transfer}. c_{2,3}! \text{prepaid}. \\
&\quad c_{1,2}? \text{confirm}. c_{2,3}! \text{payment}. \mathbf{0}
\end{aligned}$$

As explained before, the session is established via shared channels, i.e. session channels, one of which is auc. The prefix $\overline{\text{auc}}_{[2..3]}(a_{1,2}, a_{1,3})$ initiates a session with another two participants (three in total), which, in this case, are the two buyers. $\text{dTran}_{[2]}^i(b_{1,2}, b_{2,3})$, $\text{sTran}_{[2]}^i(c_{1,2}, c_{1,3}, c_{2,3})$, and $\text{epay}_{[2]}^i(d_{1,3}, d_{2,3})$ ($i \in \{1, 2\}$) are for accepting session establishment. Other prefixes are ordinary prefixes in π -calculus. We use $a_{i,j}$ to denote the channel used by the i th and j th agents in the session. For example, $a_{1,2}$ is the communicating channel between the first and second participants, which, in this case, are the broker and buyer₁. The recursive structure in P_{broker} corresponds to the recursive state of the auction protocol informally described above.

The behaviours of the two buyers in the two stages of the protocol follow. Let $j \in \{1, 2\}$.

$$\begin{aligned}
P_{\text{buyer}_j} &\stackrel{\text{def}}{=} \text{auc}_{[j+1]}(a_{1,2}, a_{1,3}). a_{1,j+1}! \text{bid}. [\text{rec } X_1](a_{1,j+1}? \text{quote}. a_{1,j+1}! \text{bid}. X \\
&\quad + a_{1,j+1}? \text{invoice}. P'_{\text{buyer}_j} + a_{1,j+1}? \text{quote}. [\text{rec } X_2] \\
&\quad (a_{1,j+1}! \text{bid}. (a_{1,j+1}? \text{invoice}. P'_{\text{buyer}_j} + a_{1,j+1}? \text{quote}. X_2)) \\
P'_{\text{buyer}_j} &\stackrel{\text{def}}{=} \overline{\text{dTran}}_{[2..3]}^j(b_{1,3}, b_{2,3}). b_{1,3}! \text{payment}. b_{1,3}? \text{order}. \mathbf{0} + \\
&\quad \overline{\text{sTran}}_{[2..3]}^j(c_{1,2}, c_{1,3}, c_{2,3}). \overline{\text{epay}}_{[2..3]}(d_{1,3}, d_{2,3}). d_{1,3}! \text{amount}. \\
&\quad d_{2,3}? \text{transfer}. c_{1,3}? \text{order}. c_{1,2}! \text{confirm}. \mathbf{0}
\end{aligned}$$

The seller and the bank only take part in the second part of the protocol, and their behaviours follow.

$$\begin{aligned}
P_{\text{seller}} &\stackrel{\text{def}}{=} \sum_{i \in \{1,2\}} (\text{dTran}_{[3]}^i(b_{1,3}, b_{2,3}). b_{2,3}? \text{price}. b_{1,3}? \text{payment}. b_{1,3}! \text{order}. \mathbf{0} \\
&\quad + \text{sTran}_{[3]}^i(c_{1,2}, c_{1,3}, c_{2,3}). c_{2,3}? \text{prepaid}. c_{1,3}! \text{order}. c_{1,3}! \text{payment}. \mathbf{0}) \\
P_{\text{bank}} &\stackrel{\text{def}}{=} \text{epay}_{[3]}(d_{1,3}, d_{2,3}). d_{1,3}? \text{amount}. d_{2,3}! \text{transfer}. \mathbf{0}
\end{aligned}$$

Fig. 4 Abstract syntax of sessions

$S, T ::= \langle p, q : v \rangle \rightarrow S$	communication		end	termination
$\langle \tilde{p} : S \rangle \{T\}$	establishment		$S; T$	concatenation
\mathbf{t}	type variable		$S \oplus T$	union
$\mu \mathbf{t}.S$	recursion		$S \otimes T$	product

The interactions of five processes, i.e. P_{broker} , P_{buyer_1} , P_{buyer_2} , P_{seller} , and P_{bank} according the operational semantics should follow the presented scenario.

3 Two-level Session Types

3.1 Syntax and Semantics

We provide the general syntax of *session types* or *sessions*, and then define the two kinds of sessions studied in the present paper, i.e. communicating and integrating sessions.

The general syntax is provided in Figure 4. $\langle p, q : v \rangle \rightarrow S$ is a session of *communication* form, meaning that, after the agent p sends the message (type) v to the agent q , the session proceeds as S . $\langle \tilde{p} : S \rangle \{T\}$ is a session of *establishment* form, meaning that the agents \tilde{p} establish a session S which nests T . The first item in the sequence \tilde{p} refers to the participant that initiates S . We call $\langle p, q : v \rangle$ and $\langle \tilde{p} : S \rangle$ *event prefixes*. $S; T$ is the *concatenation* of S and T , $S \oplus T$ is their *union*, and $S \otimes T$ their *product*. \mathbf{t} is a type variable and $\mu \mathbf{t}.S$ is a recursive type and binds \mathbf{t} in S in the standard way. A session is *close* if occurrences of all variables in it are bound. **end** is a terminated session.

If T is contained in (the presentation of) S , we call T a sub-session of S . $\text{pid}(S)$ is the set of participant names in S . S is *well-formed*, if (1) for each sub-session $\langle p, q : v \rangle \rightarrow T$ of S , $p \neq q$, (2) for each sub-session $\langle \tilde{p} : T \rangle \{T'\}$ of S , \tilde{p} is distinct and $|\tilde{p}| = |\text{pid}(T)|$, and (3) for each sub-session $T; T'$ of S , T does not contain \otimes . Hence, well-formedness of sessions rules out self interactions such as $\langle p, p : v \rangle$ and multiple participation of sessions such as $\langle p, p : S \rangle$. We also require the left session of a concatenated session to be single-threaded. For each S , we define a set $\text{opid}(S)$ as follows:

- $\text{opid}(\mathbf{t}) = \text{opid}(\mathbf{end}) = \emptyset$; $\text{opid}(\mu \mathbf{t}.S) = \text{opid}(S)$;
- $\text{opid}(\langle p, q : v \rangle \rightarrow S) = \{\{p, q\}\}$; $\text{opid}(\langle \tilde{p} : S \rangle \{T\}) = \{\tilde{p}\}$;
- $\text{opid}(S \oplus T) = \text{opid}(S \otimes T) = \text{opid}(S) \cup \text{opid}(T)$;
- • if $\text{opid}(S) \neq \emptyset$, then $\text{opid}(S; T) = \text{opid}(S)$; and
- • if $\text{opid}(S) = \emptyset$, then $\text{opid}(S; T) = \text{opid}(T)$.

Then, *race-free* sessions are recursively defined as follows. (1) **end** and $\text{opid}(\mathbf{t})$ are race-free; (2) if S is race-free, then so is $\mu \mathbf{t}.S$; (3) if S is race-free and $\{p, q\} \cap H \neq \emptyset$ for each $H \in \text{opid}(S)$, then $\langle p, q : v \rangle \rightarrow S$ is race-free for any v ; (4) if S, H are race-free, then $\langle \tilde{p} : S \rangle \{T\}$ is race-free; (5) if S, T are race-free and

Fig. 5 Session structural congruence

$(S \oplus S') \oplus S'' \equiv S \oplus (S' \oplus S'')$	$S \oplus S' \equiv S' \oplus S$	$S \oplus \text{end} \equiv S$
$(S \otimes S') \otimes S'' \equiv S \otimes (S' \otimes S'')$	$S \otimes S' \equiv S' \otimes S$	$S \otimes \text{end} \equiv S$
$(S; S'); S'' \equiv S; (S'; S'')$	$S; \text{end} \equiv S$	$\text{end}; S \equiv S$
$\mu t. S \equiv S \text{ if } t \notin \text{fv}(S)$	$S \equiv T \text{ if } S =_\alpha T$	

$H \cap H' \neq \emptyset$ for each $H \in \text{opid}(S)$, $H' \in \text{opid}(T)$, then $S \oplus T$ is race-free; (6) if S, T are race-free then $S \otimes T$ is race-free; (7) if T is race-free and $\text{pid}(S) = \emptyset$, then $S; T$ is race-free; and (8) if S, T are race-free, $\text{pid}(S) \neq \emptyset$, and $\text{pid}(S') \cap H \neq \emptyset$ for each $H \in \text{opid}(T)$ and each subsession S' of S , then $S; T$ is race-free. Let \tilde{p} be a distinct sequence and $|\tilde{p}| = \text{pid}(S)$. We use $S(\tilde{p})$ to denote the simultaneous substitution of \tilde{p} for $\text{pid}(S)$ in S .

In the following, we restrict the general syntax of sessions and define two special kinds of sessions studied in the present paper.

Definition 1 (Communicating sessions) *The syntax of communicating sessions (B, B') contains rules in Figure 4 except the establishment (i.e. $\langle \tilde{p} : S \rangle \{T\}$).*

For simplicity, for a given communicating session B , we let $\text{pid}(B)$ be a sequence of consecutive integral numbers from 1. *Auction*, *DTransaction*, *STransaction*, and *EPay* in Sect. 3.2 below are communicating sessions. However, when writing $B(\tilde{p})$, \tilde{p} is not necessarily a sequence of consecutive integrals.

Definition 2 (Integrating sessions) *The syntactic rule of restricted establishment is a restricted one of the establishment in Figure 4: $\langle \tilde{p} : B \rangle \{T\}$ where T is a general session and B is a communicating session. The syntax of integrating sessions (A, A') contains the restricted establishment and constructions in Figure 4 except the establishment and communication (i.e. $\langle p, q : v \rangle \rightarrow S$ in Figure 4).*

Proto in Sect. 3.2 below is an integrating session. We use $\langle \tilde{p} : S \rangle$ to abbreviate $\langle \tilde{p} : S \rangle \{\text{end}\}$. In the sequel, we assume that *the communicating and integrating sessions under consideration are well-formed and race-free*. Well-formedness requirement is due to syntactic legitimacy. We additionally require sessions to be race-free, because otherwise their process-level counterparts (obtained by the projection method described below) may contain race-conditions. Due to space limitations, we leave the detailed discussion for future work.

The structural congruence of sessions is the smallest congruent relation containing the equations presented in Figure 5. These laws of structural congruence have a strong correspondence to those for processes in Figure 2. Because the equi-recursive equation $\mu t. S \equiv S\{\mu t. S/t\}$ is left out, the session structural congruence is also decidable.

The operational semantics of sessions are presented in Figure 6, where we use λ to denote $p, q : v$ or $\tilde{p} : B$. [S-COM] describes the ordinary message-based

Fig. 6 Session operational semantics

$$\begin{array}{c}
\text{[S-COM]} \quad \langle p, q : v \rangle \rightarrow B \xrightarrow{p,q:v} B \quad \text{[S-SESS]} \quad \langle \tilde{p} : B \rangle \{A\} \xrightarrow{\tilde{p}:B} A \otimes B \langle \tilde{p} \rangle \\
\text{[S-TIMES]} \quad \frac{S \xrightarrow{\lambda} S'}{S \otimes T \xrightarrow{\lambda} S' \otimes T} \quad \text{[S-SUM]} \quad \frac{S \xrightarrow{\lambda} S'}{S \oplus T \xrightarrow{\lambda} S'} \quad \text{[S-CON]} \quad \frac{S \xrightarrow{\lambda} S'}{S; T \xrightarrow{\lambda} S'; T} \\
\text{[S-REC]} \quad \frac{S \xrightarrow{\lambda} S'}{\mu \mathbf{t}. S \xrightarrow{\lambda} S' \{ \mu \mathbf{t}. S / \mathbf{t} \}} \quad \text{[S-EQ]} \quad \frac{S \equiv T \quad T \xrightarrow{\lambda} T' \quad T' \equiv S'}{S \xrightarrow{\lambda} S'}
\end{array}$$

communications between two participants. [S-SESS] is for the session establishment and nesting; when a session is established, it runs interleavingly with its nested session. Other semantic rules are standard. [S-TIMES], [S-SUM], and [S-CON] handle the session production, summation, and concatenation, respectively. Recursive sessions are dealt with by [S-REC] and session equivalence by [S-EQ].

3.2 Examples of Sessions

We use the syntax of two-level sessions to formulate our business protocol. *Proto* is the session at the integrating level whilst the remaining four are at the communicating level. Their intuitive explanations have already been given in the last part of Sect. 2. We can also find a correspondence between *Proto* and its rough description in Introduction. \oplus is the multiple case of \oplus .

$$\begin{aligned}
Proto &\stackrel{\text{def}}{=} \langle \text{broker}, \text{buyer}_1, \text{buyer}_2 : Auction \rangle \{ \}; \bigoplus_{i \in \{1,2\}} \langle \text{buyer}_i, \\
&\quad \text{broker}, \text{seller} : DTransaction \rangle \{ \} \oplus \langle \text{buyer}_i, \text{broker}, \\
&\quad \text{seller} : STransaction \rangle \{ \langle \text{buyer}_i, \text{broker}, \text{bank} : EPay \rangle \{ \} \} \\
Auction &\stackrel{\text{def}}{=} \bigoplus_{i \in \{2,3\}} \langle i, 1 : \text{bid} \rangle \rightarrow \mu \mathbf{t}. \langle 1, 5 - i : \text{quote} \rangle \rightarrow \\
&\quad (\langle 1, i : \text{invoice} \rangle \rightarrow \mathbf{end} \oplus \langle 5 - i, 1 : \text{bid} \rangle \rightarrow \langle 1, i : \text{quote} \rangle \\
&\quad \rightarrow (\langle i, 1 : \text{bid} \rangle \rightarrow \mathbf{t} \oplus \langle 1, 5 - i : \text{invoice} \rangle \rightarrow \mathbf{end})) \\
DTransaction &\stackrel{\text{def}}{=} \langle 2, 3 : \text{price} \rangle \rightarrow \langle 1, 3 : \text{payment} \rangle \rightarrow \langle 3, 1 : \text{order} \rangle \rightarrow \mathbf{end} \\
STransaction &\stackrel{\text{def}}{=} \langle 2, 3 : \text{prepaid} \rangle \rightarrow \langle 3, 1 : \text{order} \rangle \rightarrow \langle 1, 2 : \text{confirm} \rangle \\
&\quad \rightarrow \langle 2, 3 : \text{payment} \rangle \rightarrow \mathbf{end} \\
EPay &\stackrel{\text{def}}{=} \langle 1, 3 : \text{amount} \rangle \rightarrow \langle 3, 2 : \text{transfer} \rangle \rightarrow \mathbf{end}
\end{aligned}$$

A comparison of the above session formulation and the behavioural formulation of the five agents in Sect. 2 leads us to see three advantages of session modularisation and integration. First, sessions characterise the interactions between processes globally, facilitating the prevention of deadlocks and race conditions. Also, sessions are free of channels. Finally, following the principle of separation of

concerns, communicating sessions partition protocols into independent modules whilst integrating sessions assemble communicating sessions at an adequately abstract level. The last aspect is unique to our theory and its merits are two-fold: it fits the natural understanding of the protocol and the gradual procedure of protocol formulation.

3.3 Role Projection

Session roles or *roles* refer to behaviours of participants acting in sessions. In other words, roles are the local description of sessions for participants. Formally, roles are represented as abstract processes. The goal of this sub-section is to develop mechanisms to project communicating and integrating sessions into their roles. The projection forms a basis for the type system developed later.

We first deal with the projection of communicating sessions. First, we mark *each* occurrence of *each* event prefix in a given session by a unique channel name. Then, we map the given session into processes according to the following rules:

- $\text{end}|r = \mathbf{0}, \mathbf{t}|r = X_{\mathbf{t}}, (\mu \mathbf{t}.B)|r = [\text{rec } X_{\mathbf{t}}](B|r),$
- $(\langle p, q : v \rangle \rightarrow B)|r =$

$$\begin{cases} c!v.(B|r) & \text{if } r = p \\ c?v.(B|r) & \text{if } r = q \\ B|r & \text{if } r \neq p \neq q \end{cases}$$

where the leftmost occurrence of $\langle p, q : v \rangle$ in $\langle p, q : v \rangle \rightarrow B$ is marked by c ,

- $(B; B')|r = (B|r)\{B'|r/\mathbf{0}\}, (B \oplus B')|r = B|r + B'|r, \text{ and } (B \otimes B')|r = B|r \mid B'|r.$

After the first two steps, we obtain a set of processes such that the *message flow* between them at the runtime (according to the operational semantics) is *deterministic* (and so channel interference is avoided). We say the message flow between the P_1 to P_m deterministic if $P_1 \mid \dots \mid P_m$ is deterministic. However, the number of channels used in sessions may be large. In the third step, we apply a channel substitution to the set of roles to optimise channel usage. The definition of the channel substitution is subject to practical considerations. For example, one may let two agents use the same channel to communicate, just as we did for the processes of five agents in the protocol example. The channel substitution is *legal* as long as the message flow between the resulted processes remains deterministic. Without confusion, when writing $B|r$, we always refer to the optimised $B|r$ and call it the role of B for r . We can check that the projection is well-defined based on the well-formedness of sessions.

The projection for integrating sessions is similar, but usually the number of communicating sessions in an integrating session is not very large, therefore we omit the channel optimisation step. First, we mark each *occurrence* of the prefix in a given integrating session by a unique channel name. Then, we map the given session into processes according to the following rules:

- $\text{end}|r = \mathbf{0}, \mathbf{t}|r = X_{\mathbf{t}}, (\mu \mathbf{t}.A)|r = [\text{rec } X_{\mathbf{t}}](A|r),$

$$\begin{aligned}
& - \langle \tilde{p} : B \rangle \{A\} \upharpoonright r = \\
& \quad \begin{cases} \bar{a}_{[2..n]}(\tilde{c}).(A \upharpoonright r) & \text{if } r = \tilde{p}[1] \wedge \text{pid}(B) = |\tilde{p}| = n \wedge \\ & |\text{fc}(B \upharpoonright r)| = |\tilde{c}| \wedge \tilde{c} \cap \text{fc}(A \upharpoonright r) = \emptyset \\ a_{[k]}(\tilde{c}).(A \upharpoonright r) & \text{if } r = \tilde{p}[k] \wedge |\text{fc}(B \upharpoonright r)| = |\tilde{c}| \wedge \tilde{c} \cap \text{fc}(A \upharpoonright r) = \emptyset \\ A \upharpoonright r & \text{if } r \notin \tilde{p} \end{cases} \\
& \quad \text{where the leftmost occurrence of } \langle \tilde{p} : B \rangle \text{ in } \langle \tilde{p} : B \rangle \{A\} \text{ is marked by } a, \\
& - (A; A') \upharpoonright r = (A \upharpoonright r) \{A' \upharpoonright r / \mathbf{0}\}, (A \oplus A') \upharpoonright r = A \upharpoonright r + A' \upharpoonright r, \text{ and } (A \otimes A') \upharpoonright r = A \upharpoonright r \mid A' \upharpoonright r.
\end{aligned}$$

The process $A \upharpoonright r$ is the role of A for r .

The projection is completely automated for integrating sessions, but it presupposes a legal channel substitution for communicating sessions to optimise the channel usage.

Examples of roles The following set of processes contains roles of *Proto* for five agents (the first five) and roles of all four communicating sessions for the broker (the last four). Let $j \in \{1, 2\}$.

$$\begin{aligned}
R_{\text{broker}}^{\text{all}} & \stackrel{\text{def}}{=} \overline{\text{auc}}_{[2..3]}(a_{1,2}, a_{1,3}). \sum_{i \in \{1,2\}} (\text{dTran}_{[2]}^i(b_{1,3}, b_{2,3}). \mathbf{0} + \\
& \quad \text{sTran}_{[2]}^i(c_{1,2}, c_{1,3}, c_{2,3}). \text{epay}_{[2]}^i(d_{1,3}, d_{2,3}). \mathbf{0}) \\
R_{\text{buyer}_j}^{\text{all}} & \stackrel{\text{def}}{=} \text{auc}_{[j+1]}(a_{1,2}, a_{1,3}). (\overline{\text{dTran}}_{[2..3]}^j(b_{1,3}, b_{2,3}). \mathbf{0} + \\
& \quad \overline{\text{sTran}}_{[2..3]}^j(c_{1,2}, c_{1,3}, c_{2,3}). \overline{\text{epay}}_{[2..3]}^j(d_{1,3}, d_{2,3}). \mathbf{0}) \\
R_{\text{seller}}^{\text{all}} & \stackrel{\text{def}}{=} \sum_{i \in \{1,2\}} (\text{dTran}_{[3]}^i(b_{1,3}, b_{2,3}). \mathbf{0} + \text{sTran}_{[3]}^j(c_{1,2}, c_{1,3}, c_{2,3}). \mathbf{0}) \\
& \quad \overline{\text{sTran}}_{[2..3]}^{i-1}(c_{1,2}, c_{1,3}, c_{2,3}). \overline{\text{epay}}_{[2..3]}^{i-1}(d_{1,3}, d_{2,3}). \mathbf{0}) \\
R_{\text{bank}}^{\text{all}} & \stackrel{\text{def}}{=} \sum_{i \in \{1,2\}} \text{epay}_{[3]}^i(d_{1,3}, d_{2,3}). \mathbf{0} \\
R_{\text{broker}}^{\text{auc}} & \stackrel{\text{def}}{=} \sum_{i \in \{1,2\}} (a_{1,i+1}?\text{bid}. [\text{rec } X_1](a_{1,4-i}!\text{quote}. (a_{1,i+1}!\text{invoice}. \mathbf{0}. + \\
& \quad a_{1,4-i}?\text{bid}. a_{1,i+1}!\text{quote}. (a_{1,i+1}?\text{bid}. X_1 + a_{1,4-i}!\text{invoice}. \mathbf{0})))) \\
R_{\text{broker}}^{\text{sTran}} & \stackrel{\text{def}}{=} c_{2,3}!\text{prepaid}. c_{1,2}?\text{confirm}. c_{2,3}!\text{payment}. \mathbf{0} \\
R_{\text{broker}}^{\text{dTran}} & \stackrel{\text{def}}{=} b_{2,3}!\text{price}. \mathbf{0} \quad R_{\text{broker}}^{\text{epay}} \stackrel{\text{def}}{=} d_{2,3}?\text{transfer}. \mathbf{0}
\end{aligned}$$

4 Type Discipline for Sessions

4.1 Type System

The purpose of the type system below is to efficiently type processes so that the ‘illegal’ runtime behaviours of processes are prevented by static type checking. The type system is based on the role projection developed earlier.

Fig. 7 Typing rules

$\Gamma \vdash \mathbf{0} \triangleright \mathbf{0} \quad \Gamma, a \triangleright B \vdash a \triangleright B$	[T-NIL], [T-CH]
$\Gamma, X \vdash X \triangleright X \text{ or } \Gamma, X \vdash \mathbf{0} \circ \tilde{c} : X$	[T-VAR]
$\frac{\Gamma \vdash a \triangleright B \quad \Gamma \vdash P \triangleright R \circ \tilde{c} : B \setminus 1\langle \tilde{c} \rangle, \Delta \quad \text{pid}(B) = [1, n]}{\Gamma \vdash \bar{a}_{[2..n]}(\tilde{c}).P \triangleright \bar{a}_{[2..n]}(\tilde{c}).R \circ \Delta}$	[T-INV]
$\frac{\Gamma \vdash a \triangleright B \quad \Gamma \vdash P \triangleright R \circ \tilde{c} : B \setminus i\langle \tilde{c} \rangle, \Delta \quad 2 \leq i \in \text{pid}(B)}{\Gamma \vdash a_{[i]}(\tilde{c}).P \triangleright a_{[i]}(\tilde{c}).R \circ \Delta}$	[T-ACC]
$\frac{\Gamma \vdash P \triangleright R \circ \Delta \quad \tilde{c} \cap (\text{dom}(\Gamma) \cup \text{ch}(\Delta)) = \emptyset}{\Gamma \vdash P \triangleright R \circ \tilde{c} : \mathbf{0}, \Delta}$	[T-TML]
$\frac{\Gamma \vdash P \triangleright R \circ \Delta \quad \tilde{c} \cap (\text{dom}(\Gamma) \cup \text{ch}(\Delta)) = \emptyset}{\Gamma \vdash P \triangleright R \circ \Delta, \tilde{c} : \mathbf{0}}$	[T-TMR]
$\frac{\Gamma \vdash P \triangleright R \circ \tilde{c} : Q, \Delta \quad b \in \tilde{c}}{\Gamma \vdash b \S v.P \triangleright R \circ \tilde{c} : b \S v.Q, \Delta} \quad \text{where } \S \in \{!, ?\}$	[T-SR]
$\frac{\Gamma \vdash P \triangleright R \circ \Delta \quad \Gamma \vdash P' \triangleright R' \circ \Delta' \quad \Delta \asymp \Delta'}{\Gamma \vdash P \mid P' \triangleright R \mid R' \circ \Delta \mid \Delta'}$	[T-COM]
$\frac{\Gamma \vdash P \triangleright R \circ \Delta \quad \Gamma \vdash P' \triangleright R' \circ \Delta' \quad \Delta \asymp \Delta'}{\Gamma \vdash P + P' \triangleright R \sqcup R' \circ \Delta \sqcup \Delta'}$	[T-SUM]
$\frac{\Gamma, X \vdash P \triangleright R \circ \tilde{c}_1 : Q_1, \dots, \tilde{c}_n : Q_n}{\Gamma \vdash [\text{rec } X]P \triangleright R^{[X]} \circ \tilde{c}_1 : Q_1^{[X]}, \dots, \tilde{c}_n : Q_n^{[X]}}$	[T-REC]
$\frac{\Gamma \vdash P \triangleright R \circ \Delta \quad R \equiv R' \quad \Delta \equiv \Delta'}{\Gamma \vdash P \triangleright R' \circ \Delta'}$	[T-EQ]
$\frac{\Gamma \vdash P \triangleright R \circ \tilde{c} : Q, \Delta \quad b \in \tilde{c}}{\Gamma \vdash (\nu b)P \triangleright R \circ \Delta, \tilde{c} \setminus b : Q, \Delta'}$	[T-HID]
$\frac{\Gamma \vdash P \triangleright R \circ \Delta \quad b \notin \text{dom}(\Gamma) \cup \text{ch}(\Delta)}{\Gamma \vdash (\nu b)P \triangleright R \circ \Delta}$	[T-VEI]
$\frac{\Gamma \vdash P \triangleright R \circ \Delta}{\Gamma \vdash l : P \triangleright R \circ \Delta}$	[T-LAB]

We define the following syntax:

$$\Gamma ::= \emptyset \mid \Gamma, a \triangleright S \mid \Gamma, X \quad \Delta ::= \{\tilde{c}_i : Q_i\}_{i \in I}$$

A *type environment* Γ is a function that assigns sessions to some channels (session channels) and *typing* to session variables. A typing is of the form $R \circ \Delta$, where R , called a *session typing*, is projected from an integrating session, and Δ , called a *channel typing*, is a sequence of processes labelled by disjoint channel sequences. The domain of Γ is a set of channels or variables it acts on. If its domain contains channel names only, we say Γ is *pure*. Re-ordering of items in a type environment Γ is permitted, but forbidden in a channel typing Δ .

The *type judgement* $\Gamma \vdash P \triangleright R \circ \Delta$ reads ‘the typing of P is $R \circ \Delta$ under Γ .’ If $\Delta = \epsilon$, we write $\Gamma \vdash P \triangleright R$. Formally, type judgements are defined by the typing rules in Figures 7, which are explained later. We also say P is *typed* or *typable* by Γ if there is a typing of P under Γ . A few auxiliary definitions are given. $|\Delta|$ is the length of Δ and $\Delta[i]$ is the i th item of Δ where $1 \leq i \leq |\Delta|$. Δ and Δ'

are *compatible*, denoted $\Delta \asymp \Delta'$, if $|\Delta| = |\Delta'|$ and $\Delta[i]$ and $\Delta'[i]$ have the same labelling sequence of channels for each $1 \leq i \leq |\Delta|$. Let $\Delta = \tilde{c}_1 : Q_1, \dots, \tilde{c}_n : Q_n$ and $\Delta' = \tilde{c}_1 : Q'_1, \dots, \tilde{c}_n : Q'_n$. $\Delta \equiv \Delta'$ if $Q_i \equiv Q'_i$ for each $1 \leq i \leq n$. Let $\Delta \sqcup \Delta' = \tilde{c}_1 : Q_1 \sqcup Q'_1, \dots, \tilde{c}_n : Q_n \sqcup Q'_n$. $\text{ch}(\Delta)$ is the set of all labelling channels in Δ .

[T-INV] and [T-ACC] are for session invitation and acceptance, and [T-SR] for the ordinary communication. [T-TML] and [T-TMR] are needed because $\Delta \asymp \Delta'$ is used in the pre-conditions of [T-COM] and [T-SUM]. By [T-VAR], the type variables happen in either the main (i.e. integrating) session or a single communicating session. [T-REC] handles the recursive construction where $P^{[X]}$ is defined in Sect. 2. [T-EQ] is necessary to make the current type system expressive enough but also bring in the infinity of typing for processes. For restricted processes e.g. $(\nu a)P$, if $a \in \text{fv}(P)$, then it is dealt with by [T-HID]; otherwise, by [T-VEI]. [T-LAB] absorbs the labelling in the typing derivation. [T-NIL], [T-CH] are standard.

We construct a (pure) type environment for the five agents in the business protocol and establish type judgements for them.

Proposition 1 *Let $\Gamma_{\text{prt}} = \text{auc} \triangleright \text{Auction}$, $\text{dTran} \triangleright \text{DTran}$, $\text{sTran} \triangleright \text{STran}$, $\text{epay} \triangleright \text{EPay}$. We have that $\Gamma_{\text{prt}} \vdash P_{\text{broker}} \triangleright R_{\text{broker}}^{\text{all}}$, $\Gamma_{\text{prt}} \vdash P_{\text{buyer}_i} \triangleright R_{\text{buyer}_i}^{\text{all}}$, $\Gamma_{\text{prt}} \vdash P_{\text{seller}} \triangleright R_{\text{seller}}^{\text{all}}$, and $\Gamma_{\text{prt}} \vdash P_{\text{bank}} \triangleright R_{\text{bank}}^{\text{all}}$.*

4.2 Properties of Typing

We study several key properties of the type system. The decidability of type inference comes first.

Theorem 1 *Given a process P and a type environment Γ , it is decidable whether there exist R, Δ such that $\Gamma \vdash P \triangleright R \circ \Delta$. If there exist, then there is an algorithm to construct such a pair.*

The proof of Theorem 1 is by computing a so-called *principal* typing for a given process under some type environment. A principal typing is a particular typing for a process such that the process has the principal typing if and only if it is typable. A standard type checking algorithm can be constructed to (attempt to) compute the principal typing for each process, and the termination of the algorithm is guaranteed by the decidability of the structural congruence for processes (cf. Lemma 1).

To present the following three properties of the type system, we put forward an auxiliary definition: for a channel typing $\Delta = \tilde{c}_1 : Q_1, \dots, \tilde{c}_n : Q_n$, let $\lceil \Delta \rceil$ be the multiple parallel-composition process $Q_1 \mid \dots \mid Q_n$. In general, $\Delta \equiv \Delta'$ is strictly stronger than $\lceil \Delta \rceil \equiv \lceil \Delta' \rceil$.

The Subject Congruence Theorem below implies that if P is typable and $P \equiv Q$ then Q is also typable and their typing have a certain structural relation.

Theorem 2 (Subject congruence) *If $\Gamma \vdash P \triangleright R \circ \Delta$ and $P \equiv P'$, then there exist R', Δ' such that $\Gamma \vdash P' \triangleright R' \circ \Delta'$, $R \equiv R'$ and $\lceil \Delta \rceil \equiv \lceil \Delta' \rceil$.*

The Subjection Reduction Theorem states that the typability of a process is preserved in an ‘expected’ way during its evolvement. The theorem rules out the standard type errors. For example, there is no Γ such that $\Gamma \vdash \bar{a}_{[2..n]}(\tilde{c}).P_1 \mid a_{[n+1]}(\tilde{c}).P_2 \mid P_3$ or $\Gamma \vdash a_{[k]}(\tilde{c}).Q_1 \mid a_{[l]}(\tilde{c}').Q_2$ where $|\tilde{c}| \neq |\tilde{c}'|$.

Theorem 3 (Subject reduction) *If $\Gamma \vdash P \triangleright R \circ \Delta$ and $P \xrightarrow{\alpha} P'$, then $\Gamma \vdash P' \triangleright R' \circ \Delta'$ for some R', Δ' satisfying the following conditions:*

1. If $\alpha = a\S v$ where $\S \in \{!, ?\}$ then $R \succ R'$ and $[\Delta] \xrightarrow{a\S v} [\Delta']$,
2. If $\alpha = \bar{a}_{[2..n]}(\tilde{c})$ and $\Gamma \vdash a \triangleright B$ then $R \xrightarrow{\bar{a}_{[2..n]}(\tilde{c})} R'$ and $B \downarrow 1\langle \tilde{c} \rangle \mid [\Delta] \succ [\Delta']$,
3. If $\alpha = a_{[k]}(\tilde{c})$ and $\Gamma \vdash a \triangleright B$ then $R \xrightarrow{a_{[k]}(\tilde{c})} R'$ and $B \downarrow k\langle \tilde{c} \rangle \mid [\Delta] \succ [\Delta']$,
4. If $\alpha = \tau$ then
 - (a) either $R \succ R'$ and $[\Delta] \xrightarrow{\tau} [\Delta']$,
 - (b) or $R \xrightarrow{\tau} R'$ and $B \downarrow 1\langle \tilde{c} \rangle \mid \dots \mid B \downarrow n\langle \tilde{c} \rangle \mid [\Delta] \succ [\Delta']$ for some B, n such that $\text{pid}(B) = [1, n]$.

The last property says that the typing of a process under a type environment is unique up to a certain structural relation as in Theorem 2.

Theorem 4 (Typing uniqueness) *If $\Gamma \vdash P \triangleright R \circ \Delta$ and $\Gamma \vdash P \triangleright R' \circ \Delta'$, then $R \equiv R'$ and $[\Delta] \equiv [\Delta']$.*

5 Behavioural Analysis

In this section, we use the two-level session types to analyse interactions of distributed program components. Two system properties are dealt with: a channel safety property (also studied in the existing session type literature) and a behavioural conformance between processes and sessions. But we first show how to represent and properly type a distributed system in our formalism.

Informally, a program or a component in a distributed system is a pair of a participant name and a process. Following works in the process algebraic approach to architectural analysis, such as [1, 2, 26], we define (the architecture of) a system as a parallel composition of programs or components. Formally, we define that

Definition 3 (Systems) *A program or a component is a labelled process $r : P$, where r and P specify its name and behaviour, respectively. A system is a process of the form $\text{Sys} = r_1 : P_1 \mid \dots \mid r_n : P_n$.*

For example, the following system implements the business protocol introduced in the Introduction and formalised in Sect. 3.2.

$$\begin{aligned} \text{Sys}_e = & \text{broker} : P_{\text{broker}} \mid \text{buyer}_1 : P_{\text{buyer}_1} \mid \\ & \text{buyer}_2 : P_{\text{buyer}_2} \mid \text{seller} : P_{\text{seller}} \mid \text{bank} : P_{\text{bank}} \end{aligned}$$

A session channel a marks B in A if some occurrence of B in A is marked by a in the projection. The following definition characterises how to use session types to properly type a system.

Definition 4 (Session well-typedness) *Sys* (as defined in Def. 3) is well-typed by A_{spc} under Γ if

- $\text{pid}(A_{\text{spc}}) = \{r_1, \dots, r_n\}$,
- $\Gamma \vdash a \triangleright B$ if and only if a marks B in A_{spc} , and
- $\Gamma \vdash P_i \triangleright A_{\text{spc}} \upharpoonright r_i$ for each $1 \leq i \leq n$.

If *Sys* is well-typed by A_{spc} , we call A_{spc} a session for *Sys*. In general, well-typedness is strictly stronger than typability. In other words, if *Sys* is well-typed by A_{spc} under Γ then $\Gamma \vdash \text{Sys} \triangleright A_{\text{spc}}$; but the other direction does not necessarily hold. Also, we observe that if *Sys* is well-typed by some session and *Sys* (as a process) is a close then *Sys* is well-typed by some pure session.

The channel safety property below says that channel interference is prevented at the runtime of the system.

Definition 5 (Channel privacy) *The communicating channels in Sys are private if the following holds: if $\text{Sys} \xrightarrow{\tau}_* (\nu \tilde{b})(r_1 : P_1 \mid \dots \mid r_n : P_n)$ and $c \S v \in \text{act}(P_i)$ where $\S \in \{!, ?\}$, then there exists a unique r_j such that $r_i \neq r_j$ and $c \in \text{fc}(P_j)$.*

The channel privacy of a system is a consequence of well-typedness by a session specification, as the following theorem demonstrates.

Theorem 5 *If Sys is well-typed by A under Γ , then the communicating channels in Sys are private.*

Informally, the theorem is guaranteed by the determinism of message flow in the projected roles and the creation of fresh channels in the session establishment.

Session conformance says that the runtime interactions of the system conform to its session specification.

Definition 6 (Session conformance) *P conforms to S, if there is a relation \mathcal{R} of processes and sessions such that if $\langle P, S \rangle \in \mathcal{R}$ then the following conditions hold:*

- If $P \equiv (\nu \tilde{b})(p : Q_1 \mid q : Q_2 \mid R)$, $Q_1 \xrightarrow{a!v} Q'_1$ and $Q_2 \xrightarrow{a?v} Q'_2$, then there exists S' such that $S \xrightarrow{p,q:v} S'$ and $\langle P', S' \rangle \in \mathcal{R}$ where $P' \equiv (\nu \tilde{b})(p : Q'_1 \mid q : Q'_2 \mid R)$;
- If $P \equiv (\nu \tilde{b})(p_1 : Q_1 \mid \dots \mid p_m : Q_m \mid R)$, $Q_1 \xrightarrow{\bar{a}[2..m](\tilde{c})} Q'_1$ and $Q_i \xrightarrow{a[i](\tilde{c})} Q'_i$ for all $2 \leq i \leq m$, then there exist B, S' such that $\Gamma \vdash a \triangleright B$, $S \xrightarrow{p_1, \dots, p_m : B} S'$ and $\langle P', S' \rangle \in \mathcal{R}$, where $P' \equiv (\nu \tilde{b})((\nu \tilde{c})(p_1 : Q'_1 \mid \dots \mid p_m : Q'_m) \mid R)$.

An alternative explanation of session conformance is behavioural refinement, because Def. 6 actually defines a behavioural stimulation relation between sessions and processes. We observe that if P conforms to S and $P \equiv P'$, then P' conforms to S . The following theorem confirms that session conformance of the system is also a consequence of well-typedness by a session specification.

Theorem 6 *If Sys is well-typed by A_{spc} then Sys conforms to A_{spc} .*

By what we have established so far, we have the following two properties for the system Sys_e :

Proposition 2 (1) *The communicating channels are private in Sys_e .* (2) *The behaviour of Sys_e conforms to its session specification Proto.*

6 Process Slicing

A type inference algorithm computes a typing, if possible, for a process under a type environment (cf. Theorem 1). However, in the real-life cases, the developers have the session specification in the first place and then implement it, so they need to check whether a process is typable by the given session specification. A straightforward method to solve this type checking problem consists of two steps: to verify whether $\Gamma \vdash P \triangleright A \upharpoonright r$ for some $r : P, A, \Gamma$, we first compute $\Gamma \vdash P \triangleright A_P \upharpoonright r$ by a type inference algorithm and then check whether $A_P \equiv A$. This algorithm is efficient, pre-supposing we have an efficient type inference algorithm.

However, there is a drawback in the above algorithm: if $\Gamma \vdash P \triangleright A \upharpoonright r$ does not hold, the algorithm does not tell which session or sessions it violates. Since the session specification is modularised, it is desirable to know the violated session or sessions. In the following, we propose an algorithm based on process slicing to improve the type checking. Informally, the key idea of the algorithm is to decompose a process into parts and compare each part with a role projected from a corresponding session.

Suppose each session channel in P is typed by Γ , namely, contained in the domain of Γ . The algorithm consists of two steps. The first step is the process slicing. Because the hiding and labelling operators are unnecessary for processes as the initial (not runtime) behaviours of programs or components, we assume that P is free of these two operators. We call P^{χ_M} the main slice of P and the main slicing function χ_M is formally defined as follows.

$$\begin{aligned} \mathbf{0}^{\chi_M} &= \mathbf{0} & X^{\chi_M} &= X & ([\text{rec } X]P)^{\chi_M} &= [\text{rec } X](P)^{\chi_M}, \\ (\pi.P)^{\chi_M} &= \begin{cases} \pi.(P)^{\chi_M} & \text{if } \pi \text{ is a session action,} \\ P^{\chi_M} & \text{otherwise;} \end{cases} \\ (P \star Q)^{\chi_M} &= P^{\chi_M} \star Q^{\chi_M} & \text{where } \star &\text{ is } | \text{ or } +. \end{aligned}$$

We call $P^{\chi_{\tilde{c}}}$ the \tilde{c} -slice of P , and the slicing function $\chi_{\tilde{c}}$, which are parametric on \tilde{c} , is defined below.

$$\begin{aligned} \mathbf{0}^{\chi_{\tilde{c}}} &= \mathbf{0} & X^{\chi_{\tilde{c}}} &= X & ([\text{rec } X]P)^{\chi_{\tilde{c}}} &= [\text{rec } X](P)^{\chi_{\tilde{c}}}, \\ (\pi.P)^{\chi_{\tilde{c}}} &= \begin{cases} P^{\chi_{\tilde{c}}} & \text{if } \text{fc}(\pi) \notin \tilde{c}, \\ \pi.(P)^{\chi_{\tilde{c}}} & \text{otherwise;} \end{cases} \\ (P \star Q)^{\chi_{\tilde{c}}} &= P^{\chi_{\tilde{c}}} \star Q^{\chi_{\tilde{c}}} & \text{where } \star &\text{ is } | \text{ or } +. \end{aligned}$$

After computing the slices of a process, we check whether each slice is structurally congruent to a corresponding role. Specifically, we verify if $A \upharpoonright r \equiv (\pi.P)^{\chi_M}$, $B \upharpoonright 1\langle \tilde{c} \rangle \equiv P^{\chi_{\tilde{c}}}$, and $B \upharpoonright k\langle \tilde{c} \rangle \equiv P^{\chi_{\tilde{c}}}$, where $\Gamma \vdash a \triangleright B$.

By by our bound name convention, a name is not bound twice and does not have free *and* bound occurrences simultaneously in a process. The following theorem says that if P is typed by $A \upharpoonright r$ under Γ then the slicing of P ‘coincides’ with the role projection of A .

Theorem 7 (Slicing-projection correspondence) *If $\Gamma \vdash P \triangleright A \upharpoonright r \langle \tilde{a} \rangle$ then the following three conditions hold:*

- $A \upharpoonright r \langle \tilde{a} \rangle \equiv P^{\chi_M}$,
- if $\tilde{a}_{[2..n]}(\tilde{c}) \in \text{act}(P)$ and $\Gamma \vdash a \triangleright B$, then $B \upharpoonright 1 \langle \tilde{c} \rangle \equiv P^{\chi_{\tilde{c}}}$,
- if $a_{[k]}(\tilde{c}) \in \text{act}(P)$ and $\Gamma \vdash a \triangleright B$, then $B \upharpoonright k \langle \tilde{c} \rangle \equiv P^{\chi_{\tilde{c}}}$.

Based on the above theorem, the correctness of the process slicing algorithm for the type checking is established. However, the method is not complete: the other direction of the theorem does not hold, as witnessed by the following counter-example. Thereby, the coincidence of role projection and process slicing does not entail the typability, and a technical implication is that the process slicing method cannot replace the type system in Sect. 4.1.

Proposition 3 *Let $B_1 = \langle p, q : v_1 \rangle \rightarrow \langle p, q : u_1 \rangle \rightarrow \text{end}$, $B_2 = \langle q, p : v_2 \rangle \rightarrow \langle q, p : u_2 \rangle \rightarrow \text{end}$, $A_0 = B_1; B_2$, $\Gamma_0 \vdash a_i \triangleright B_i$ where $i \in \{1, 2\}$, and $P_1 = a_{[2]}^1(c_1). c_1 ? v_1. a_{[2..2]}^2(c_2). c_2 ! v_2. c_1 ? u_1. c_2 ! u_2. \text{end}$. With a suitable role projection of B_1 and B_2 , we have that A_0, P_1 and Γ_0 satisfy the three conditions in Theorem 7 but not $\Gamma_0 \vdash P_1 \triangleright A_0 \upharpoonright p \langle a_1, a_2 \rangle$.*

7 Related Work

Session type theories Our work is rooted in the forgoing theories of session types, especially the global description of interactions and multiparty sessions. Carbone *et al.* [4] presented two calculi to describe the communication behaviours from the global and local perspectives, respectively, and several principles to establish a sound and complete projection of the former to the latter. Some of the ideas behind the syntactic restrictions that we set up for the two-level sessions are related to their projection principles. The process calculus in the present paper is from Honda *et al.* [15], in which the authors extended the traditional binary session types to the multiparty asynchronous context and solved several technical channels (as the result of the loss of two-party duality and the asynchrony) such that several fundamental properties of the session type discipline also hold by linearity analysis. The syntax for the calculus is abstract (e.g. messages are treated as message types) and does not contain some syntactic features that are considered as essential to session type theories (e.g. the distinction of internal and external choices and message-based branching behaviours, as argued by Castagnal and Padovani [5]). Our intention is to focus on the two-level separation of session syntax and minimise the side techniques when studying relevant properties. We leave the work on enriching the syntax of session and calculi alike according to the existing session type theories in the future.

The subsequent work on session types witnesses a trend of increment on the expressive power to characterise richer conversation structures. For example, Deniérou and Yoshida [8] extended the multiparty session types to accommodate the runtime change of session participants, i.e. the joining or leaving of participants, after a session is initiated. The same authors [28] introduced a finite recursive type constructors into the multiparty session types to express a wide range of processes whose specification structures are parameterised whilst keeping the type checking for the resulting type system decidable. To improve protocol modularisation of session types, Demangeon and Honda [7] introduced a way to define abstract nested protocols independent of their host protocols such that the host protocols can call the nested ones by passing them arguments such as values, roles, and even (names of) other protocols. In these studies, the enrichment of the session type construction leads to the increment of syntactic primitives in the process calculi. In contrast, the separation of two-level sessions in our work does not complicate the syntax of the calculus. An interesting point is to compare the concept of nested protocols by Demangeon and Honda [7] with that in the present paper. Their protocol calling is comparable to the procedure calling in the sequential programming, in which the exact position of the involvement must be specified to make sense of the main program. Our protocol nesting is more general in the sense that ‘being nested by’ just means ‘occurring within’. Also, in our work, the meaning of the host protocol is complete with or without its nested protocol(s).

Padovani [22] proposed a backward approach to session types, in which session types are defined as projected fragments of processes. More specifically, a process is sliced as per channels it uses and session types are a type approximation of the channel-sliced fragments of the process. There are two connecting points between his work and ours: first, both make use of process slicing, in spite of different purposes; second, both (and [4]) investigate sessions semantically.

Session types as architectural connection The idea of viewing sessions as a behavioural approximation of processes comes from process algebraic analysis of software architectural connection. Architectural connection deals with the interactions of components which contrast to the local computations of components. Allan and Garlan [1] argued for the merits of implementing architectural connection in a special class of components called connectors. They formulated connector types based on the process algebra CSP [11] and analysed the protocol compatibility issues related to components and connectors. Following their approach, the present authors [25][26] proposed formal languages and methods to improve the architectural analysis. But these works assume the co-ordination of connectors for components and, hence, only handle the connector-based architectural styles. Bernardo *et al.* [2] distinguished the connector-based and non-connector-based styles, but their analytic techniques for the latter are based on the notion of ‘inter-operability’ of a process against others, which skirts around the problem. Multiparty session types offer a solution to overcome the restriction by describing the component interactions globally without using connectors. To employ multiparty session types to analyse architectural connection, we need

to be concerned with the behavioural compatibility (defined as session conformance in the present paper) between component computations (processes) and their expected interactions (session types).

8 Conclusions

We address the problem of session integration in protocol specification and develop a theory of two-level synchronous multiparty session types, in which session integration is separated from session communication. As of the technical results, we develop a new type system and study its key properties. We also analyse a channel safety property and a behavioural relation between processes and sessions, and present a process slicing method to improve the type checking.

We outline several interesting directions for further studies. First, we are working on the analysis of more behavioural properties of distributed computing systems in the novel session type theory. For example, behavioural refinement is too weak in some sense, and we want to establish a relation between behavioural refinement and equivalence between processes and sessions. Deadlock-freedom and liveness of processes are also important properties to be studied. The challenge is to properly revise the set of typing rules so that the satisfaction of some natural properties by the sessions entails the satisfaction of those behavioural properties. Second, we also expect to enrich the syntax of our process calculus according to existing session type studies. Third, the process slicing method is not complete with respect to the type system and, therefore, one research question revolves around finding a complete method to facilitate the type checking. Finally, we are also interested in leveraging session types as a theoretic tool for software architectural analysis.

References

1. R. Allen and D. Garlan, “A formal basis for architectural connection,” *ACM TSEM*, 6:213-249, 1997.
2. M. Bernardo, P. Ciancarini, and L. Donatiello, “Architecting families of software systems with process algebras,” *ACM TSEM*, 11(4), 2002.
3. N. Bhatti, M. Hiltunen, R. Schlichting, and W. Chiu, “Coyote: A system for constructing fine-grain configurable communication services,” *ACM Trans. on Computer Systems*, 16(4):321, 1998.
4. M. Carbone, K. Honda, and N. Yoshida, “Structured communication-centred programming for web service,” *ESOP’07*, 2007.
5. G. Castagna, L. Padovani, “Contracts for Mobile Processes,” *CONCUR’09*, 2009.
6. L. de Alfaro and T. Henzinger, “Interface automata,” *ESEC/FSE-9*, 2001.
7. R. Demangeon and K. Honda, “Nested Protocols in Session Types,” *CONCUR’12* 2012.
8. P. Deniérou, N. Yoshida, “Dynamic multirole session types,” *POPL’11*, 2011.
9. J. Engelfriet, and T. Gelsema, “The decidability of structural congruence for replication restricted pi-calculus processes,” *LIACS Technical Report*, 2004.

10. S. Gay and M. Hole, “Types and Subtypes for Client-Server Interactions,” *ESOP’99*, 1999.
11. C.A.R. Hoare, “Communicating sequential processes,” *Communication of ACM*, 1978.
12. M. Hennessy, *A Distributed Pi-Calculus*, Cambridge University Press, 2007.
13. K. Honda, A. Mukhamedov, G. Brown, T. Chen, and N. Yoshida, “Scribbling Interactions with a Formal Foundation,” *ICDCIT’11* 2011.
14. K. Honda, V. Vasconcelos, and M. Kubo, “Language primitives and type disciplines for structured communication-based programming,” *ESOP’98*, 1998
15. K. Honda, N. Yoshida, M. Carbone, “Multiparty asynchronous session types,” *POPL’08*, 2008.
16. P. Inverardi, A. Wolf, and D. Yankelevich, “Static checking of system behaviors using derived component assumptions,” *ACM TSEM*, 11:386-426, 2000.
17. M. Leclercq, V. Quema, and J. Stefani, “DREAM: a component framework for the construction of resource-aware, configurable MOMs,” *IEEE Distributed Systems Online*, 6(9), 2005.
18. M. Lienhardt, A. Schmitt, and J. Stefani, “Typing communicating component assemblages,” *GPCE’08*, 2008.
19. J. Magee, N. Dulay, S. Eisenbach, and J. Kramer, “Specifying distributed software architectures,” *ESEC’95*, 1995.
20. H. Miranda, A. Pinto, and L. Rodrigues, “Appia: A flexible protocol kernel supporting multiple coordinated channels,” *ICDCS’01*, 2001.
21. F. Oquendo, “ π -ADL: An architecture description language based on the higher order typed π -Calculus for specifying dynamic and mobile software architectures,” *ACM Software Engineering Notes* 29(3), 2004.
22. L. Padovani, “On projecting processes into session types,” *Mathematical Structures in Computer Science*, Vol. 22, Special Issue 02, pp. 237-89, Cambridge University Press, 2012
23. J. Pérez, L. Caires, F. Pfenning, and B. Toninho, “Linear logical relations for session-Based concurrency,” *ESOP’12*, 2012: 539-558.
24. D. Sangiorgi and D. Walker, *The π -calculus: A Theory of Mobile Processes*, Cambridge University Press, 2001.
25. G. Su, M. Ying, and C. Zhang, “An ADL-approach to specifying and analyzing centralized-mode architectural connection,” *ECISA’10*, 2010.
26. G. Su, M. Ying, and C. Zhang, “Semantic analysis of component-aspect dynamism for connector-based architecture styles,” *WICSA/ECISA’12*, 2012.
27. J. Sifakis, “A Framework for Component-based Construction,” *SEFM’05*, 2005.
28. N. Yoshida, P. Deniérou, A. Bejleri, and R. Hu. Parameterised multiparty session types. In *Foundations of Software Science and Computational Structures, FoS-SaCS’10*, volume 6014 of LNCS, pages 128C145, 2010.
29. D. Yellin and R. Strom, “Protocol specifications and components adaptors”, *ACM Transactions on Programming Languages and Systems*, 19(2), 292C333, 1997.

Appendix

A Complete Set of Roles for The Protocol Example

Roles of Proto for the five agents Let $j \in \{1, 2\}$.

$$\begin{aligned}
R_{\text{broker}}^{\text{all}} &\stackrel{\text{def}}{=} \overline{\text{auc}}_{[2..3]}(a_{1,2}, a_{1,3}). \sum_{i \in \{1,2\}} (\text{dTran}_{[2]}^i(b_{1,3}, b_{2,3}). \mathbf{0} + \\
&\quad \text{sTran}_{[2]}^i(c_{1,2}, c_{1,3}, c_{2,3}). \text{epay}_{[2]}^i(d_{1,3}, d_{2,3}). \mathbf{0}) \\
R_{\text{buyer}_j}^{\text{all}} &\stackrel{\text{def}}{=} \text{auc}_{[j+1]}(a_{1,2}, a_{1,3}). (\overline{\text{dTran}}_{[2..3]}^j(b_{1,3}, b_{2,3}). \mathbf{0} + \\
&\quad \overline{\text{sTran}}_{[2..3]}^j(c_{1,2}, c_{1,3}, c_{2,3}). \overline{\text{epay}}_{[2..3]}^j(d_{1,3}, d_{2,3}). \mathbf{0}) \\
R_{\text{seller}}^{\text{all}} &\stackrel{\text{def}}{=} \sum_{i \in \{1,2\}} (\text{dTran}_{[3]}^i(b_{1,3}, b_{2,3}). \mathbf{0} + \text{sTran}_{[3]}^i(c_{1,2}, c_{1,3}, c_{2,3}). \mathbf{0}) \\
&\quad \overline{\text{sTran}}_{[2..3]}^{i-1}(c_{1,2}, c_{1,3}, c_{2,3}). \overline{\text{epay}}_{[2..3]}^{i-1}(d_{1,3}, d_{2,3}). \mathbf{0}) \\
R_{\text{bank}}^{\text{all}} &\stackrel{\text{def}}{=} \sum_{i \in \{1,2\}} \text{epay}_{[3]}^i(d_{1,3}, d_{2,3}). \mathbf{0}
\end{aligned}$$

Roles of the four communicating sessions for the broker

$$\begin{aligned}
R_{\text{broker}}^{\text{auc}} &\stackrel{\text{def}}{=} \sum_{i \in \{1,2\}} (a_{1,i+1}?\text{bid}. [\text{rec } X_1](a_{1,4-i}!\text{quote}. (a_{1,i+1}!\text{invoice}. \mathbf{0} + \\
&\quad a_{1,4-i}?\text{bid}. a_{1,i+1}!\text{quote}. (a_{1,i+1}?\text{bid}. X_1 + a_{1,4-i}!\text{invoice}. \mathbf{0})))) \\
R_{\text{broker}}^{\text{sTran}} &\stackrel{\text{def}}{=} c_{2,3}!\text{prepaid}. c_{1,2}?\text{confirm}. c_{2,3}!\text{payment}. \mathbf{0} \\
R_{\text{broker}}^{\text{dTran}} &\stackrel{\text{def}}{=} b_{2,3}!\text{price}. \mathbf{0} \quad R_{\text{broker}}^{\text{epay}} \stackrel{\text{def}}{=} d_{2,3}?\text{transfer}. \mathbf{0}
\end{aligned}$$

Roles of the four communicating sessions for the buyers Let $j \in \{1, 2\}$.

$$\begin{aligned}
R_{\text{buyer}_j}^{\text{auc}} &\stackrel{\text{def}}{=} a_{1,j+1}!\text{bid}. [\text{rec } X_1](a_{1,j+1}?\text{quote}. a_{1,j+1}!\text{bid}. X + \\
&\quad a_{1,j+1}?\text{invoice}. \mathbf{0}) + a_{1,j+1}?\text{quote}. \\
&\quad [\text{rec } X_2](a_{1,j+1}!\text{bid}. (a_{1,j+1}?\text{invoice}. \mathbf{0} + a_{1,j+1}?\text{quote}. X_2)) \\
R_{\text{buyer}_j}^{\text{dTran}} &\stackrel{\text{def}}{=} b_{1,3}!\text{payment}. b_{1,3}?\text{order}. \mathbf{0} \\
R_{\text{buyer}_j}^{\text{sTran}} &\stackrel{\text{def}}{=} c_{1,3}?\text{order}. c_{1,2}!\text{confirm}. \mathbf{0} \quad R_{\text{buyer}_j}^{\text{epay}} \stackrel{\text{def}}{=} d_{1,3}!\text{amount}. \mathbf{0}
\end{aligned}$$

Roles of DTransaction and DTransaction for the seller

$$\begin{aligned}
R_{\text{seller}}^{\text{dTran}} &\stackrel{\text{def}}{=} b_{2,3}?\text{price}. b_{1,3}?\text{payment}. b_{1,3}!\text{order}. \mathbf{0} \\
R_{\text{seller}}^{\text{sTran}} &\stackrel{\text{def}}{=} c_{2,3}?\text{prepaid}. c_{1,3}!\text{order}. c_{1,3}!\text{payment}. \mathbf{0}
\end{aligned}$$

The role of EPay for the bank

$$R_{\text{bank}}^{\text{epay}} \stackrel{\text{def}}{=} d_{1,3}?\text{amount}. d_{2,3}!\text{transfer}. \mathbf{0}$$

B Derivation of $\Gamma_{\text{prt}} \vdash P_{\text{broker}} \triangleright R_{\text{broker}}^{\text{all}}$

This part of the appendix is dedicated to detailing a derivation of the type judgement $\Gamma_{\text{prt}} \vdash P_{\text{broker}} \triangleright R_{\text{broker}}^{\text{all}}$ in Proposition 1. Derivations of other type judgements in Proposition 1 can be constructed in a similar way.

1. by [T-NIL][T-TML]:

$$\Gamma_{\text{prt}} \vdash \mathbf{0} \triangleright \mathbf{0} \circ (c_{1,2}, c_{1,3}, c_{2,3}) : \mathbf{0}$$

2. by [T-SR]:

$$\Gamma_{\text{prt}} \vdash R_{\text{broker}}^{\text{sTran}} \triangleright \mathbf{0} \circ (c_{1,2}, c_{1,3}, c_{2,3}) : R_{\text{broker}}^{\text{sTran}}$$

3. by [T-TML][T-SR]:

$$\Gamma_{\text{prt}} \vdash d_{1,3}?\text{transfer}. R_{\text{broker}}^{\text{sTran}} \triangleright \mathbf{0} \circ (d_{1,3}, d_{2,3}) : R_{\text{broker}}^{\text{epay}}, (c_{1,2}, c_{1,3}, c_{2,3}) : R_{\text{broker}}^{\text{sTran}}$$

4. let $i \in \{1, 2\}$ and

$$P_1^i \stackrel{\text{def}}{=} \text{sTran}_{[2]}^i(c_{1,2}, c_{1,3}, c_{2,3}). \text{epay}_{[2]}^i(d_{1,3}, d_{2,3}). d_{1,3}?\text{transfer}. R_{\text{broker}}^{\text{sTran}}$$

5. by [T-CH][T-ACC]:

$$\Gamma_{\text{prt}} \vdash P_1^i \triangleright \text{sTran}_{[2]}^i(c_{1,2}, c_{1,3}, c_{2,3}). \text{epay}_{[2]}^i(d_{1,3}, d_{2,3}). \mathbf{0}$$

6. by [T-NIL][T-TML][T-SR][T-CH][T-ACC]:

$$\Gamma_{\text{prt}} \vdash \text{dTran}_{[2]}^i(b_{1,2}, b_{2,3}). b_{2,3}!\text{price}. \mathbf{0} \triangleright \text{dTran}_{[2]}^i(b_{1,2}, b_{2,3}). \mathbf{0}$$

7. let

$$P_2^i = \text{dTran}_{[2]}^i(b_{1,2}, b_{2,3}). \mathbf{0} + \text{sTran}_{[2]}^i(c_{1,2}, c_{1,3}, c_{2,3}). \text{epay}_{[2]}^i(d_{1,3}, d_{2,3}). \mathbf{0}$$

8. by (5)(6)[T-SUM]:

$$\Gamma_{\text{prt}} \vdash P_{\text{broker}}^i \triangleright P_2^i \quad \Gamma_{\text{prt}} \vdash P_{\text{broker}}^{3-i} \triangleright P_2^{3-i}$$

9. by [T-TML][T-SR]:

$$\Gamma_{\text{prt}} \vdash a_{1,4-i}!\text{invoice}. P_{\text{broker}}^{3-i} \triangleright P_2^{3-i} : (a_{1,2}, a_{1,3}) : a_{1,4-i}!\text{invoice}. \mathbf{0}$$

10. by [T-VAR][T-TML][T-SR]:

$$\Gamma_{\text{prt}} \vdash a_{1,i+i}?\text{bid}. X \triangleright \mathbf{0} \circ (a_{1,2}, a_{1,3}) : a_{1,i}?\text{bid}. X$$

11. let

$$P_3^i = a_{1,4-i}?\text{bid}. a_{1,i+1}!\text{quote}. (a_{1,i+1}?\text{bid}. X + a_{1,4-i}!\text{invoice}. \mathbf{0})$$

12. by (9)(10)[T-SUM][T-SR][T-EQ]:

$$\Gamma_{\text{prt}} \vdash P_3^i \{P_{\text{broker}}^{3-i}/\mathbf{0}\} \triangleright P_2^{3-i} \circ (a_{1,2}, a_{1,3}) : P_3^i$$

13. by [T-TML][T-SR]:

$$\Gamma_{\text{prt}} \vdash a_{1,i+1}!\text{invoice}. P_{\text{broker}}^i \triangleright P_2^i \circ (a_{1,2}, a_{1,3}) : a_{1,i+1}!\text{invoice}. \mathbf{0}$$

14. let

$$\begin{aligned} P_4^i &= a_{1,4-i}!\text{quote}. (a_{1,i+1}!\text{invoice}. P_{\text{broker}}^i + P_3^i \{P_{\text{broker}}^{3-i}/\mathbf{0}\}) \\ P_5^i &= a_{1,4-i}!\text{quote}. (a_{1,i+1}!\text{invoice}. \mathbf{0} + P_3^i) \end{aligned}$$

15. by (12)(13)[T-SUM][T-SR]:

$$\Gamma_{\text{prt}} \vdash P_4^i \triangleright P_2^i + P_2^{3-i} \circ a_{1,4-i}!\text{quote}. (a_{1,i+1}!\text{invoice}. \mathbf{0} + P_3^i)$$

16. by [T-REC][T-SR]:

$$\Gamma_{\text{prt}} \vdash a_{1,i+1}?\text{bid}. [\text{rec } X] P_4^i \triangleright P_2^i + P_2^{3-i} \circ a_{1,i+1}?\text{bid}. [\text{rec } X] P_5^i$$

17. by (16)(4)[T-SUM] ($P_2^1 + P_2^{3-1} \equiv P_2^2 + P_2^{3-2}$):

$$\Gamma_{\text{prt}} \vdash \sum_{i \in \{1,2\}} P_4^i \triangleright P_2^1 + P_2^2 \circ R_{\text{broker}}^{\text{auc}}$$

18. by [T-INV]:

$$\Gamma_{\text{prt}} \vdash P_{\text{broker}} \triangleright R_{\text{broker}}^{\text{all}}$$

This finishes the derivation.

C More Examples

We present two more examples to show the utility of our two-level session types in expressing the scenario-based specification of systems.

Client-server system The first one is a client-server system, which consists of one client, two servers and a configurator. The client attempts to make requests to the servers and the configurator co-ordinates the client and two servers so that the client can only call the available server(s). Both servers have two states: they are either in the normal working order or preparing to update their data bases and shut down the service temporarily. The servers inform the configurator of their states in their conversations. Before the client calls the servers, the configurator tells them whether the servers are ready to take requests.

The following is a formulation of the session specification in our two-level session types, where C is the client, S_1, S_2 are two servers, F is the configurator, $CSystem$ is an integrating session type, and $Control$, $Initi$ and $Service$ are communicating session types.

$$\begin{aligned}
CSystem &\stackrel{\text{def}}{=} \langle F, S_1, S_2 : Control \rangle \{ \} \\
&\quad \otimes \mu \mathbf{t}. (\bigoplus_{i \in \{1,2\}} \langle C, F : Initi \rangle \{ \}; \langle C, S_i : Service \rangle \{ \}; \mathbf{t}) \\
Control &\stackrel{\text{def}}{=} \mu \mathbf{t}. (\bigoplus_{j \in \{2,3\}} \langle j, 1 : \text{update} \rangle \rightarrow \langle j, 1 : \text{ready} \rangle \rightarrow \mathbf{t}) \\
Initi &\stackrel{\text{def}}{=} \bigoplus_{k \in \{1,2\}} \langle 1, 2 : \text{ping}_k \rangle \rightarrow (\langle 1, 2 : \text{yes} \rightarrow \text{end} \rangle \oplus \langle 1, 2 : \text{no} \rangle \rightarrow \text{end}) \\
Service &\stackrel{\text{def}}{=} \langle 1, 2 : \text{request} \rangle \rightarrow \langle 2, 1 : \text{return} \rangle \rightarrow \text{end}
\end{aligned}$$

The formulation captures the intuitive and coarse-grained understanding of the conversations between the four components. First, the conversations consists of three parts, represented by three communicating sessions. Second, the relationship of these sessions is described by $CSystem$, revealing the most essential design decisions of the system. For example, $Service$ happens after $Initi$ and together they form a recursive session. $Control$ is also recursive and proceeds independent of the other two communicating sessions. Of course, many design details are to be worked out in the later development stage. For example, if the configurator replies ‘no’ to the client’s pinging action in $Initi$, then the client is not allowed to initiate $Service$. Also, the messages received by the configurator in $Control$ should affect its replies to the client’s pinging action in $Initi$.

Quote request The second example is a quote request protocol which is modified and simplified from the one in [28]. The protocol involves three agents, i.e. a buyer, a supplier, and a manufacturer, and consists of two parts: the first part is a conversation between the buyer and the supplier, in which the price of some item or good is negotiated; the second part, which is nested within in the first part, is for the supplier to confirm the price with the manufacturer.

As before, the formulate consists of one integrating session and several (here is two) communicating sessions. B stands the buyer, S the supplier, and M the manufacturer.

$$\begin{aligned}
QuoteReq &\stackrel{\text{def}}{=} \langle B, S : Negotn \rangle \{ \langle S, M : Confirm \rangle \{ \} \} \\
Negotn &\stackrel{\text{def}}{=} \langle 1, 2 : \text{item} \rangle \rightarrow \langle 2, 1 : \text{quote} \rangle \rightarrow (\langle 1, 2 : \text{accepted} \rangle \rightarrow \text{end} \oplus \\
&\quad \langle 1, 2 : \text{newquote} \rangle \rightarrow (\langle 2, 1 : \text{accepted} \rangle \rightarrow \text{end} \oplus \\
&\quad \langle 2, 1 : \text{rejected} \rangle \rightarrow \text{end})) \\
Confirm &\stackrel{\text{def}}{=} \langle 1, 2 : \text{quote} \rangle \rightarrow (\langle 1, 2 : \text{yes} \rightarrow \text{end} \rangle \oplus \langle 1, 2 : \text{no} \rangle \rightarrow \text{end})
\end{aligned}$$

This example shows the necessity to distinguish protocol calling and protocol nesting (c.f. discussions in Sect. 7). Because, as far as the protocol is concerned,

it suffices to indicate the nesting relationship between *Negotn* and *Confirm*. Without specifying the nesting position of *Confirm* in *Negotn*, *Negotn* describes a complete conversation between the buyer and the supplier.

D Proof Details

Proof of Theorem 1

Proof. The proof of is a standard proof of decidability of type inference. Because of [T-EQ], a process has infinite many typing, but we show that we can compute a ‘principal’ typing for each process such that the process has a ‘principal’ typing if and only if it is typable.

First, for each P , we compute a set $\text{sub}_|(P)$ (resp. $\text{sub}_\sqcup(P)$) which is the smallest set such that

1. if $\langle P_1, P_2 \rangle \in \text{sub}_|(P)$ (resp. $\text{sub}_\sqcup(P)$) then $P_1 \mid P_2 \equiv P$ (resp. $P_1 \sqcup P_2 \equiv P$), and
2. if $Q_1 \mid Q_2 \equiv P$ (resp. $Q_1 \sqcup Q_2 \equiv P$) then there are P_1, P_2 such that $P_1 \equiv Q_1$, $P_2 \equiv Q_2$ and $\langle P_1, P_2 \rangle \in \text{sub}_|(P)$ (resp. $\text{sub}_\sqcup(P)$).

The two sets are decidable because \equiv is decidable (Lemma 1). $\text{sub}_|(\Delta)$ is defined as follows: $\langle \Delta_1, \Delta_2 \rangle \in \text{sub}_|(\Delta)$ if and only if $|\Delta_1| = |\Delta_2| = |\Delta|$ and, for each $1 \leq i \leq |\Delta|$, $\langle P_1^i, P_2^i \rangle \in \text{sub}_|(P^i)$ where $\Delta_1[i] = \tilde{c} : P_1^i$, $\Delta_2[i] = \tilde{c} : P_2^i$ and $P^i = \tilde{c} : \Delta[i]$ for some \tilde{c} . $\text{sub}_\sqcup(\Delta)$ is defined similarly. Note that if $\langle \Delta_1, \Delta_2 \rangle \in \text{sub}_|(\Delta)$ (resp. $\text{sub}_\sqcup(\Delta)$), then $\Delta_1 \preceq \Delta_2$.

A *principal* typing of P under Γ is a typing derived by the rules in Figures 7 except [T-EQ], and plus the following two rules:

$$\frac{\Gamma \vdash P \triangleright R \circ \Delta \quad \Gamma \vdash P' \triangleright R' \circ \Delta' \quad \langle R, R' \rangle \in \text{sub}_|(R'') \quad \langle \Delta, \Delta' \rangle \in \text{set}_|(\Delta'')}{\Gamma \vdash P \mid P' \triangleright R'' \circ \Delta''} \quad [\text{T-COM+}]$$

$$\frac{\Gamma \vdash P \triangleright R \circ \Delta \quad \Gamma \vdash P' \triangleright R' \circ \Delta' \quad \langle R, R' \rangle \in \text{sub}_\sqcup(R'') \quad \langle \Delta, \Delta' \rangle \in \text{set}_\sqcup(\Delta'')}{\Gamma \vdash P + P' \triangleright R'' \circ \Delta''} \quad [\text{T-SUM+}]$$

We have the following lemma:

Lemma 2 $\Gamma \vdash P \triangleright R \circ \Delta$ if and only if P has a principal typing under Γ .

The right-to-left direction of the lemma is obvious. For the other direction, we suppose $\Gamma \vdash P \triangleright R \circ \Delta$. In the derivative procedure, if commutative and associative laws for \mid and $+$ are applied in [T-EQ], we have the same derivation by the additional two derived rules, and whenever other structural laws in Figure 2 are applied in [T-EQ], we just omit them. In this manner, we will obtain a principal typing for P . Therefore, the type inference of the type system is decidable.

Note that if $R' \circ \Delta'$, say, is the principal typing of P , then by Theorem 4 (to be proved) $R \equiv R'$ and $[\Delta] = [\Delta']$.

Proof of Theorem 2

Proof. Suppose $\Gamma \vdash P$ and $P \equiv P'$. The proof is by induction on the derivation of $P \equiv P'$. The proof is divided into two parts. First, we show that the each rule in Figure 2 and its symmetric form respect the above theorem. Here we detail one of the most tricky rules:

$$(\nu a)P_1 \mid P_2 \equiv (\nu a)(P_1 \mid P_2) \text{ if } a \notin \text{fc}(P_2)$$

(1) We first suppose $P = (\nu a)P_1 \mid P_2$, $\Gamma \vdash P \triangleright R \circ \Delta$ and $a \notin \text{fc}(P_2)$. Because $\Gamma \vdash P \triangleright R \circ \Delta$ is derived by [T-COM] and possibly by [T-EQ], [T-TML] and/or [T-TMR] (for one or more times), it can be verified that $\Gamma \vdash (\nu a)P_1 \triangleright R_1 \circ \Delta_1$, $\Gamma \vdash P_2 \triangleright R_2 \circ \Delta_2$, $\Delta_1 \asymp \Delta_2$, $R \equiv R_1 \mid R_2$, and $[\Delta] \equiv [\Delta_1 \mid \Delta_2]$ for some $R_1, R_2, \Delta_1, \Delta_2$. Here we have two possibilities. (1.1) Suppose $a \in \text{ch}(\Delta_1)$. Thus, $\Gamma \vdash (\nu a)P_1 \triangleright R_1 \circ \Delta_1$ is derived by [T-HID] and possibly by [T-EQ][T-TML][T-TMR], and, we have that $\Gamma \vdash P_1 \triangleright R_1 \circ \Delta'_1$, $\tilde{c} : Q$, Δ'_1 , $a \in \tilde{c}$, and $\Delta_1 \equiv \tilde{c}_1 : \mathbf{0}, \dots, \tilde{c}_m : \mathbf{0}, \Delta'_1, \tilde{c} \backslash a : Q, \Delta'_1, \tilde{c}_{m+1} : \mathbf{0}, \dots, \tilde{c}_{m+n} : \mathbf{0}$. No matter $a \in \bigcup_{i=1}^{m+n} \tilde{c}_i$ or not, we can rewrite the processes of type derivations for P_1 and P_2 to obtain type judgements $\Gamma \vdash P_1 \triangleright R_1 \circ \Delta_3$ and $\Gamma \vdash P_2 \triangleright R_2 \circ \Delta_2$ such that $\Delta_3 \asymp \Delta_4$ and $[\Delta_3 \mid \Delta_4] = [\Delta_1 \mid \Delta_2]$ (when applying [T-TML] or [T-TMR] to prefix \tilde{b} for some \tilde{b} , we prefix \tilde{b}/a or some \tilde{b}' such that $\tilde{b}' \backslash a = \tilde{b}$ instead). Note that the rewritten derivations are based on $a \in \text{fc}(P_2)$ and the channel assumption (cf. Sect. 2). Then, we apply [T-HID] to type $(\nu a)(P_1 \mid P_2)$ and obtain the desired result. (1.2) Suppose $a \notin \text{ch}(\Delta_1)$. Thus, $\Gamma \vdash (\nu a)P_1 \triangleright R_1 \circ \Delta_1$ is derived by [T-VEI] and possibly by [T-EQ][T-TML][T-TMR], and we have that $\Gamma \vdash P_1 \triangleright R_1 \circ \Delta'_1$ and $\Delta_1 \equiv \tilde{c}_1 : \mathbf{0}, \dots, \tilde{c}_m : \mathbf{0}, \Delta'_1, \tilde{c}_{m+1} : \mathbf{0}, \dots, \tilde{c}_{m+n} : \mathbf{0}$. Similarly, we rewrite the type derivation for P_2 and obtain $\Gamma \vdash P_2 \triangleright R_2 \circ \Delta'_2$ such that $\Delta_1 \asymp \Delta'_2$ and $[\Delta'_2] \equiv [\Delta_2]$. Then, apply [T-VEI] to $\Gamma \vdash P_2 \triangleright R_2 \circ \Delta'_2$ and obtain the desired result. (2) Then, we suppose $P = (\nu a)(P_1 \mid P_2)$, $\Gamma \vdash P \triangleright R \circ \Delta$ and $a \notin \text{fc}(P_2)$. The treatment is similar to the first case.

The second part of the proof is to show that the laws of congruence respect the theorem. We choose to deal with the following rule:

$$\frac{P_1 \equiv P_2}{[\text{rec } X]P_1 \equiv [\text{rec } X]P_2}$$

We suppose $P = [\text{rec } X]P_1$, $P' = [\text{rec } X]P_2$, $P_1 \equiv P_2$, and $\Gamma \vdash P \triangleright R \circ \Delta$ where $\Delta \equiv \tilde{c}_1 : Q_1, \dots, \tilde{c}_n : Q_n$. Because $\Gamma \vdash P \triangleright R \circ \Delta$ is derived by [T-REC] (and [T-EQ][T-TML][T-TMR], possibly), we have $\Gamma, X \vdash P_1 \triangleright R_1 \circ \tilde{c}_1 : Q'_1, \dots, \tilde{c}_n : Q'_n$ such that $R_1 \equiv R^{[X]}$ and $Q'_i \equiv Q_i^{[X]}$ for each $1 \leq i \leq n$. Since $P_1 \equiv P_2$, by induction hypotheses, $\Gamma, X \vdash P_2 \triangleright R_2 \circ \tilde{c}'_1 : Q''_1, \dots, \tilde{c}'_n : Q''_n$ for some R_2 , \tilde{c}'_i and Q''_i for each $1 \leq i \leq n$ such that $R_2 \equiv R_1$ and $Q'_1 \mid \dots \mid Q'_n \equiv Q''_1 \mid \dots \mid Q''_n$. By [T-REC], we have that $\Gamma \vdash [\text{rec } X]P_2 \triangleright R \circ \Delta'$ where $R \equiv R_2^{[X]}$ and $\Delta' \equiv \tilde{c}'_1 : Q''_1^{[X]}, \dots, \tilde{c}'_n : Q''_n^{[X]}$. Therefore, we have that $R \equiv R'$ and $[\Delta] \equiv [\Delta']$.

Proof of Theorem 3

Proof. The proof is by induction on the derivation of $P \xrightarrow{\alpha} P'$ according to rules in Figure 3 and depends on the value of α . The following only covers the most typical cases.

(1) Suppose $P = \alpha.P'$. In this case, α has three possible forms: $a\S v$, $\bar{a}_{[2..n]}(\tilde{c})$ or $a_{[k]}(\tilde{c})$ where $\S \in \{?, !\}$. First, we let $\alpha = a\S v$. Because $\Gamma \vdash P \triangleright R \circ \Delta$ is derived by [T-SR] and possibly by [T-EQ][T-TML][T-TMR] (for one or more times), we have that $\Gamma \vdash P' \triangleright R' \circ \Delta'$ for some R', Δ' such that $R \equiv R'$, $[\Delta'] \equiv [\tilde{c} : Q, \Delta'']$, and $[\Delta] \equiv [\tilde{c} : a\S v.Q, \Delta'']$ for some \tilde{c}, Q, Δ'' . We have that $[\Delta] \xrightarrow{\alpha} \succ [\Delta']$. Then, let $\alpha = \bar{a}_{[2..n]}(\tilde{c})$. Because $\Gamma \vdash P \triangleright R \circ \Delta$ is derived by [T-INV] (and possibly [T-EQ][T-TML][T-TMR]), we have that $\Gamma \vdash P' \triangleright R' \circ \Delta'$, $R \equiv \bar{a}_{[2..n]}(\tilde{c}).R'$ (thus $R' \xrightarrow{\alpha} R$) and $[\Delta] \equiv [\Delta'] \mid B!1\langle\tilde{c}\rangle$ where $\Gamma \vdash a \vdash B$. Lastly, let $\alpha = a_{[k]}(\tilde{c})$. Because $\Gamma \vdash P \triangleright R \circ \Delta$ is derived by [T-ACC] (and possibly [T-EQ][T-TML][T-TMR]), we have that $\Gamma \vdash P' \triangleright R' \circ \Delta'$, $R \equiv a_{[k]}(\tilde{c}).R'$ (thus $R' \xrightarrow{\alpha} R$), and $[\Delta] \equiv [\Delta'] \mid B!1\langle\tilde{c}\rangle$ where $\Gamma \vdash a \vdash B$ and $2 \leq k \in \text{pid}(B)$.

(2) Let $\alpha = a\S v$ and suppose $P = P_1 + P_2$ and $P \xrightarrow{\alpha} P'$ is derived from $P_1 \xrightarrow{\alpha} P'$ (the treatment is similar of it derived from $P_2 \xrightarrow{\alpha} P'$). By [T-SUM] (and possibly by [T-EQ][T-TML][T-TMR]), we have that $\Gamma \vdash P_1 \triangleright R_1 \circ \Delta_1$, $\Gamma \vdash P_2 \triangleright R_2 \circ \Delta_2$, $R \equiv R_1 \sqcup R_2$, and $[\Delta] \equiv [\Delta_1 \sqcup \Delta_2]$. By induction hypotheses, $\Gamma \vdash P_1' \triangleright R_1' \circ \Delta_1'$, $R_1 \succ R_1'$ and $[\Delta_1] \xrightarrow{\alpha} \succ [\Delta_1']$. Hence, $R \succ R'$ and $[\Delta] \xrightarrow{\alpha} \succ [\Delta']$.

(3) Let $\alpha = \tau$ and $P = P_1 \mid P_2$. Here we have two subcases. (3.1) Suppose $P \xrightarrow{\tau} P'$ is derived from $P_1 \xrightarrow{\tau} P'$ (or $P_2 \xrightarrow{\tau} P'$). The treatment for this subcase is relatively simple and similar to the last case and thus we omit it. (3.2) $P \xrightarrow{\tau} P'$ is derived from $P_1 \xrightarrow{a!v} P_1'$ and $P_2 \xrightarrow{a?v} P_2'$ and $P' = P_1' \mid P_2'$. Because $\Gamma \vdash P \triangleright R \circ \Delta$ is derived by [T-COM] (and possibly [T-EQ][T-TML][T-TMR]), we have that $\Gamma \vdash P_1 \triangleright R_1 \circ \Delta_1$ and $\Gamma \vdash P_2 \triangleright R_2 \circ \Delta_2$ for some $R_1, R_2, \Delta_1, \Delta_2$ such that $R \equiv R_1 \mid R_2$, and $[\Delta] \equiv [\Delta_1 \mid \Delta_2]$. By induction hypotheses, $\Gamma \vdash P_1' \triangleright R_1' \circ \Delta_1'$ and $\Gamma \vdash P_2' \triangleright R_2' \circ \Delta_2'$ for some $R_1', R_2', \Delta_1', \Delta_2'$ such that $R_1 \succ R_1'$, $R_2 \succ R_2'$, $[\Delta_1] \xrightarrow{a!v} \succ [\Delta_1']$ and $[\Delta_2] \xrightarrow{a?v} \succ [\Delta_2']$. Also, $\Delta_1' \prec \Delta_2'$. Hence, $\Gamma \vdash P' \triangleright R_1' \mid R_2' \circ \Delta_1' \mid \Delta_2'$, $R_1 \mid R_2 \succ R_1' \mid R_2'$ and $[\Delta_1 \mid \Delta_2] \xrightarrow{\tau} \succ [\Delta_1' \mid \Delta_2']$. (3.3) $P \xrightarrow{\tau} P'$ is derived from $P_1 \xrightarrow{\bar{a}_{[2..n]}(\tilde{c})} P_2'$ and $P_i \xrightarrow{a_{[i]}(\tilde{c})} P_i'$ for each $2 \leq i \leq n$. Suppose $\Gamma \vdash a \triangle B$. Because $\Gamma \vdash P \triangleright R \circ \Delta$ is derived by [T-COM] for $n - 1$ times (and possibly [T-EQ][T-TML][T-TMR]), we have that $\Gamma \vdash P_1 \triangleright R_1 \circ \Delta_1$ and $\Gamma \vdash P_i \triangleright R_i \circ \Delta_i$ ($2 \leq i \leq n$) for some $R_1, R_i, \Delta_1, \Delta_i$ such that $R \equiv R_1 \mid R_2 \mid \dots \mid R_n$ and $[\Delta] \equiv [\Delta_1] \mid [\Delta_2] \mid \dots \mid [\Delta_n]$. By induction hypotheses, $\Gamma \vdash P_1' \triangleright R_1' \circ \Delta_1'$ and $\Gamma \vdash P_i' \triangleright R_i' \circ \Delta_i'$ ($2 \leq i \leq n$) for some $R_1', R_i', \Delta_1', \Delta_i'$ such that $R_1 \xrightarrow{\bar{a}_{[2..n]}(\tilde{c})} R_1'$, $R_i \xrightarrow{a_{[i]}(\tilde{c})} R_i'$, $[\Delta_1'] \equiv [\Delta_1] \mid B!1\langle\tilde{c}\rangle$, and $[\Delta_i'] \equiv [\Delta_i] \mid B!i\langle\tilde{c}\rangle$ for each $2 \leq i \leq n$. Also, $\Delta_1' \prec \Delta_2' \prec \dots \prec \Delta_n'$. Therefore, $[\Delta_1' \mid \dots \mid \Delta_n'] \equiv B!1\langle\tilde{c}\rangle \mid \dots \mid B!n\langle\tilde{c}\rangle \mid [\Delta_1 \mid \dots \mid \Delta_n]$.

(4) Let $\alpha = a\S v$. Suppose $Q \equiv P$, $Q' \equiv P'$, and $P \xrightarrow{\alpha} P'$ is derived from $Q \xrightarrow{\alpha} Q'$. By Theorem 2, $\Gamma \vdash Q \triangleright R_1 \circ \Delta_1$ such that $R_1 \equiv R$ and $[\Delta_1] \equiv [\Delta]$. By induction hypotheses, $\Gamma \vdash Q' \triangleright R_1' \circ \Delta_1'$ such that $R_1 \succ R_1'$ and $[\Delta_1] \xrightarrow{\alpha} \succ [\Delta_1']$. Then, by Theorem 2 again, we have the desired result.

Proof of Theorem 4

Proof. The proof is by induction on the derivation of $\Gamma \vdash P \triangleright R \circ \Delta$ and $\Gamma \vdash P \triangleright R' \circ \Delta'$ according to rules in Figure 7. We detail two cases. (1) Suppose $P = \bar{a}_{[2..n]}(\tilde{c}).P_1$ and $\Gamma \vdash P \triangleright R \circ \Delta$ is derived by [T-INV]. Let $\Gamma \vdash a \triangleright B$ and $|\text{pid}(B)| = n$. Thus, $\Gamma \vdash P_1 \triangleright R_1 \circ \Delta_1$ for some R_1, Δ_1 such that $R = \bar{a}_{[2..n]}(\tilde{c}).R_1$ and $\Delta_1 = \tilde{c} : B \upharpoonright 1, \Delta$. Also, $\Gamma \vdash P \triangleright R' \circ \Delta'$ and possibly [T-EQ][T-TML][T-TMR] for one or more times. Thus, $\Gamma \vdash P_1 \triangleright R'_1 \circ \Delta'_1$ for some R'_1, Δ'_1 such that $R' \equiv \bar{a}_{[2..n]}(\tilde{c}).R'_1$ and $\lceil \Delta'_1 \rceil \equiv B \upharpoonright 1 \mid \lceil \Delta' \rceil$. By induction hypotheses, $R_1 \equiv R'_1$ and $\lceil \Delta_1 \rceil = \lceil \Delta'_1 \rceil$. Therefore, $R \equiv R'$ and $\lceil \Delta \rceil = \lceil \Delta' \rceil$. (2) Suppose $P = [\text{rec } X]P'$, and $\Gamma \vdash P \triangleright R \circ \Delta$ and $\Gamma \vdash P \triangleright R' \circ \Delta'$ are derived by [T-REC]. Let $\Gamma \vdash P' \triangleright R_1 \circ \tilde{c}_1 : Q_1^1, \dots, \tilde{c}_n : Q_n^1$ where $R_1^{[X]} = R$ and $\tilde{c}_1 : Q_1^{1[X]}, \dots, \tilde{c}_n : Q_n^{1[X]} = \Delta$, and $\Gamma \vdash P' \triangleright R_2 \circ \tilde{c}_1 : Q_1^2, \dots, \tilde{c}_n : Q_n^2$ where $R_2^{[X]} = R'$ and $\tilde{c}_1 : Q_1^{2[X]}, \dots, \tilde{c}_n : Q_n^{2[X]} = \Delta'$. By induction hypotheses, $R_1 \equiv R_2$ and $\lceil Q_1^1 \mid \dots \mid Q_n^1 \rceil \equiv \lceil Q_1^2 \mid \dots \mid Q_n^2 \rceil$ for each $1 \leq i \leq n$. Thus, we have $R \equiv R'$ and $\lceil \Delta \rceil \equiv \lceil \Delta' \rceil$.

Proof of Theorem 5

Proof. (Sketch) This lemma is guaranteed by the projection of sessions into roles and the generation of fresh channels in the session establishment. A formal proof is by induction on $\text{Sys} \xrightarrow{\tau}_* (\nu \tilde{a})(1 : P_1 \mid \dots \mid n : P_n)$.

Proof of Theorem 6

Proof. We first put forward two lemmas, whose proofs are by the syntax of B or A .

Lemma 3 (1) If $B \xrightarrow{p,q:v} B'$ then $B \setminus p \langle \tilde{c} \rangle \xrightarrow{b?v} B' \setminus p \langle \tilde{c}' \rangle$ and $B \setminus q \langle \tilde{c} \rangle \xrightarrow{b!v} B' \setminus q \langle \tilde{c}' \rangle$ for some $b \in \tilde{c} \supseteq \tilde{c}'$. (2) If $B \setminus p \langle \tilde{c} \rangle \xrightarrow{b?v} P$ and $B \setminus q \langle \tilde{c} \rangle \xrightarrow{b!v} Q$ then there are B', \tilde{c}' such that $B \xrightarrow{p,q:v} B'$, $P \equiv B' \setminus p \langle \tilde{c}' \rangle$, $Q \equiv B' \setminus q \langle \tilde{c}' \rangle$ and $\tilde{c}' \subseteq \tilde{c}$.

Lemma 4 Let $\tilde{p} = p_1, \dots, p_m$ and b marks B in A . (1) If $A \xrightarrow{\tilde{p}:B} A' \otimes B \langle \tilde{p} \rangle$ then $A \setminus p_1 \xrightarrow{\tilde{b}_{[2..m]}(\tilde{c})} A' \setminus p_1$ and $A \setminus p_i \xrightarrow{b_{[i]}(\tilde{c})} A' \setminus p_i$ ($2 \leq i \leq m$). (2) If $A \setminus p_1 \xrightarrow{\tilde{b}_{[2..m]}(\tilde{c})} P_1$ and $A \setminus p_i \xrightarrow{b_{[i]}(\tilde{c})} P_i$ ($2 \leq i \leq m$) then there is A' such that $P_j \equiv A' \setminus p_j$ ($1 \leq j \leq m$) and $A \xrightarrow{\tilde{p}:B} A' \otimes B \langle \tilde{p} \rangle$.

Suppose Sys is well-typed by A_{spc} . We construct an \mathcal{R} such that $\langle P, S \rangle \in \mathcal{R}$ if and only if

- $\text{Sys} \xrightarrow{\tau}_* P = (\nu \tilde{b})(1 : P_1 \mid \dots \mid n : P_n)$,
- $A_{\text{spc}} \xrightarrow{\tau}_* S = C \otimes B_1 \otimes \dots \otimes B_k$,
- for each $1 \leq i \leq n$, $\Gamma \vdash P_i \triangleright R_i \circ \tilde{c}_1 : Q_1^i, \dots, \tilde{c}_k : Q_k^i$ where
 - $C \setminus i \langle \tilde{a} \rangle \succ R_i$,
 - $B_j \setminus i \langle \tilde{c}_j \rangle \succ Q_j^i$ for each $1 \leq j \leq k$.

First, we have that $\langle \text{Sys}, A_{\text{spc}} \rangle \in \mathcal{R}$. Then, let $\langle P, S \rangle \in \mathcal{R}$ and suppose the above five induction hypotheses. Without loss of generality, we suppose (1) $P_1 \xrightarrow{b!v} P'_1$ and $P_2 \xrightarrow{b?v} P'_2$ or (2) $P_1 \xrightarrow{\bar{b}[2..m](\tilde{c}_0)} P'_1$ and $P_i \xrightarrow{b[i](\tilde{c}_0)} P'_i$ ($2 \leq i \leq m$) and b marks B_0 .

(1) Suppose $b \in \tilde{c}_j$. By Theorem 3, $Q_j^1 \xrightarrow{b!v} Q_j^{1'}$ and $Q_j^2 \xrightarrow{b?v} Q_j^{2'}$. Thus, by Lemma 3, $B_j \upharpoonright 1 \langle \tilde{c}_j \rangle \xrightarrow{b!v} B'_j \upharpoonright 1 \langle \tilde{c}'_j \rangle$ and $B_j \upharpoonright 2 \langle \tilde{c}_j \rangle \xrightarrow{b?v} B'_j \upharpoonright 2 \langle \tilde{c}'_j \rangle$ for some B'_j, \tilde{c}'_j such that $B_j \xrightarrow{1,2:v} B'_j$, $B'_j \upharpoonright 1 \langle \tilde{c}'_j \rangle \succ Q_j^{1'}$ and $B'_j \upharpoonright 2 \langle \tilde{c}'_j \rangle \succ Q_j^{2'}$. Also, $B_j \upharpoonright i \langle \tilde{c}_j \rangle = B'_j \upharpoonright i \langle \tilde{c}'_j \rangle$ for each $3 \leq i \leq n$. Therefore, let $S' = C \otimes B'_1 \otimes B_2 \dots \otimes B_k$ and $P' = (\nu \tilde{b})(1 : P'_1 \mid 2 : P'_2 \mid 3 : P_3 \mid \dots \mid P_n)$. We have that $\langle P', S' \rangle \in \mathcal{R}$. (2) By Theorem 3, $R_1 \xrightarrow{\bar{b}[2..m](\tilde{c}_0)} R'_1$ and $R_i \xrightarrow{b[i](\tilde{c}_0)} R'_i$ ($2 \leq i \leq m$). Thus, by Lemma 4, $C \upharpoonright \langle \tilde{a} \rangle \xrightarrow{\bar{b}[2..m](\tilde{c}_0)} C' \upharpoonright \langle \tilde{a} \rangle$ and $C \upharpoonright \langle \tilde{a} \rangle \xrightarrow{b[i](\tilde{c}_0)} C' \upharpoonright \langle \tilde{a} \rangle$ ($2 \leq i \leq m$) for some $C', \tilde{a}, \tilde{a}'$ such that $b \in \tilde{a} \supseteq \tilde{a}'$, $C \xrightarrow{1,\dots,m;\tilde{b}_0} C'$, $C' \upharpoonright l \langle \tilde{a}' \rangle \succ R'_l$ ($2 \leq l \leq m$). Also, $C \upharpoonright l \langle \tilde{a} \rangle \equiv C' \upharpoonright l \langle \tilde{a}' \rangle$ for each $m+1 \leq l \leq n$. Therefore, let $S' = C' \otimes B_0 \otimes B_1 \otimes \dots \otimes B_k$ and $P' = (\nu \tilde{b})(1 : P'_1 \mid \dots \mid m : P'_m \mid P_{m+1} \dots \mid P_n)$. We have that $\langle P', S' \rangle \in \mathcal{R}$.

Proof of Theorem 7

Proof. We prove a more general proposition: if $\Gamma \vdash P \triangleright R \circ \Delta$ then

- $P^{X_M} \equiv R$,
- if $\bar{a}[2..n](\tilde{c}) \in \text{act}(P)$ and $\Gamma \vdash a \triangleright B$, then $B \upharpoonright 1 \equiv P^{X_{\tilde{c}}}$,
- if $a[k](\tilde{c}) \in \text{act}(P)$ and $\Gamma \vdash a \triangleright B$, then $B \upharpoonright k \equiv P^{X_{\tilde{c}}}$,
- if $\Delta[i] = \tilde{c} : Q$, then $Q \equiv P^{X_{\tilde{c}}}$.

We observe that if the above proposition holds then Theorem 7 immediately follows. The proof of the proposition is by induction on the derivation of $\Gamma \vdash P \triangleright R \circ \Delta$ according to Figure 7. The basic cases are simple. For the non-basic cases, we choose to deal with two typical cases. (1) $P = a[2..n](\tilde{c}).P'$ and $\Gamma \vdash P \triangleright R \circ \Delta$ is derived by [T-INV]. Let $R = \bar{a}[2..n](\tilde{c}).R'$ and $\Delta = \tilde{c} : B \upharpoonright 1, \Delta'$. By induction hypotheses, $P'^{X_M} \equiv R'$ and, thus, $P^{X_M} \equiv R$. Suppose $\bar{b}[2..m](\tilde{c}') \in \text{act}(P)$, $\Gamma \vdash b \triangleright B$, and $\tilde{c}' \cap \text{ch}(\Delta) = \emptyset$. If $b = a$ (and thus $\tilde{c}' = \tilde{c}$, then by induction hypotheses and the rule [T-INV], $P'^{X_{\tilde{c}}} = B \upharpoonright 1$. If $b \neq a \dots$, by induction hypotheses, we have the same result. The case of $b[i](\tilde{c}')$ is similar. Suppose $\Delta[i] = \tilde{c}' : Q$. Then, $\tilde{c}' \cap \tilde{c} = \emptyset$ and $\Delta'[i+1] = \tilde{c} : Q$. Thus, $P^{X_{\tilde{c}'}} = P'^{X_{\tilde{c}'}}$ and by induction hypotheses $Q \equiv P^{X_{\tilde{c}'}}$. (2) $P = P_1 \mid P_2$ and $\Gamma \vdash P \triangleright R \circ \Delta$ is derived by [T-COM]. Let $R = R_1 \mid R_2$, $\Delta = \Delta_1 \mid \Delta_2$ and $\Gamma \vdash P_i \triangleright R_i \circ \Delta_i$ where $i \in \{1, 2\}$. By induction hypotheses and the rule [T-COM], we can obtain the four propositions above. (N.B. we have suppose P does not contain the hiding operator, so the rules [T-HID] and [T-VEI] are not applicable.)